行记录。

（5）继续查询 emp 表的下一行记录。

（6）重复第 2 步。



单行或多行子查询都会在外部查询值比较前进行一次值比较，而关联子查询则必须在外查询的每一行记录上进行一次值比较。如果使用合适的操作符，一个关联子查询可能是单行或者多行。

1. 使用 EXISTS 操作符

在关联子查询中可以使用 EXISTS 或 NOT EXISTS 操作符。其中，EXISTS 操作符用于检查子查询所返回的行是否存在，它可以在非关联子查询中使用，但是更常用于关联子查询。

下面编写 SQL 语句，查询 "SALES" 部门的所有员工信息，如下：

```
SQL>  SELECT empno 员工编号,ename 员工姓名,deptno 所在部门编号,sal 工资 FROM emp
  2  WHERE EXISTS
  3  (SELECT deptno FROM dept WHERE deptno=emp.deptno AND dname='SALES');

  员工编号    员工姓名     所在部门编号      工资
---------- ---------- ------------ ----------
      7521  WARD               30        1250
      7844  TURNER             30        1500
      7499  ALLEN              30        1600
      7900  JAMES              30         950
      7698  BLAKE              30        2850
      7654  MARTIN             30        1250
已选择 6 行。
```



使用 EXISTS 操作符，只是检查子查询返回的数据是否存在，因此，在子查询语句中可以不返回一列，而返回一个常量值，这样可以提高查询的性能，比如使用常量 1 来代替上述子查询 SELECT 之后的 deptno 列，查询的结果是一样的。

2. 使用 NOT EXISTS 操作符

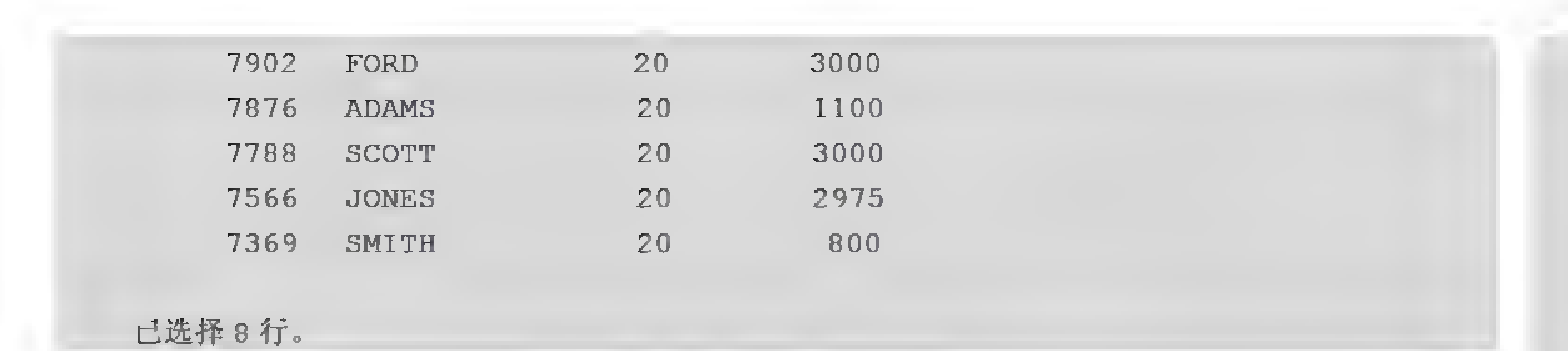
在执行的操作逻辑上，NOT EXISTS 操作符的作用与 EXISTS 操作符相反。在需要检查数据行中是否不存在子查询返回的结果时，就可以使用 NOT EXISTS。

使用 NOT EXISTS 操作符，检索不在 "SALES" 部门工作的所有员工信息，语句如下：

```
SQL>  SELECT empno 员工编号,ename 员工姓名,deptno 所在部门编号,sal 工资 FROM emp

  2  WHERE NOT EXISTS
  3  (SELECT deptno FROM dept WHERE deptno=emp.deptno AND dname='SALES');

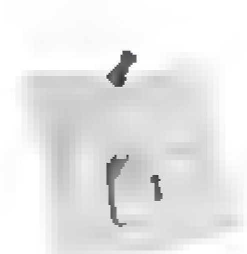
  员工编号    员工姓名     所在部门编号     工资
---------- ---------- ------------ ----------
      7934  MILLER            10       1300
      7839  KING              10       5000
      7782  CLARK             10       2450
```



```
      7902    FORD              20          3000
      7876    ADAMS             20          1100
      7788    SCOTT             20          3000
      7566    JONES             20          2975
      7369    SMITH             20           800

已选择 8 行。
```

9.6.4　网络课堂



视频教学：http://school.itzcn.com/video-vid-1178-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16726-1-1.html

9.7　复杂的查询问题

9.7.1　问题描述

我想要对 SCOTT 模式下的 emp 表和 dept 表进行操作，获取部门员工平均工资高于那些部门编号大于 10 的所有部门的员工平均工资最高值的部门号和平均工资，实在是有点复杂，这样的 SQL 语句如何编写啊？

9.7.2　解决方法

分析问题，我们可以先获取部门号大于 10 的所有部门编号，然后使用 IN 操作符获取部门号大于 10 的所有部门员工的平均工资最高值，然后再使用 HAVING 子句获取部门员工平均工资高于部门号大于 10 的所有部门员工的平均工资最高值的部门号和平均工资，完整的 SQL 语句如下：

```
SQL> SELECT deptno,AVG(sal)
  2  FROM emp
  3  GROUP BY deptno
  4  HAVING AVG(sal)>
  5  (SELECT MAX(AVG(sal)) FROM emp WHERE deptno IN (SELECT deptno FROM dept
  WHE
RE deptno>10) GROUP BY deptno);

DEPTNO      AVG(SAL)
----------  ----------
10          2916.66667
```

可以看到，这个例子非常复杂，它包含了 3 个查询：一个嵌套子查询、一个子查询和一

个外部查询，整个查询就是按照这种顺序执行的。

9.7.3 知识扩展——实现嵌套子查询

所谓嵌套子查询，是指在子查询内部使用其他子查询。嵌套子查询的嵌套层次最多为255层。大多数情况下，嵌套子查询都在外层子查询的WHERE子句中。

下面使用嵌套子查询来获取就职于“SALES”部门工作的所有员工信息：

```
SQL>  SELECT  empno 员工编号,ename 员工姓名,deptno 所在部门编号,sal 工资
  2   FROM emp
  3   WHERE empno IN
  4  (SELECT empno FROM emp WHERE deptno=
  5  (SELECT deptno FROM dept WHERE dname='SALES')
  6  );

  员工编号     员工姓名       所在部门编号     工资
---------- ---------- ------------ ----------
      7499   ALLEN              30        1600
      7521   WARD               30        1250
      7654   MARTIN             30        1250
      7698   BLAKE              30        2850
      7844   TURNER             30        1500
      7900   JAMES              30         950

已选择6行。
```

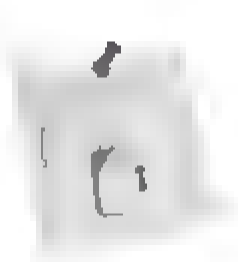
从上述的SQL语句中可以看出，这个例子是非常复杂的。它包含了3个查询：一个嵌套子查询、一个子查询和一个外部查询。整个查询就是按照这种顺序执行的。先执行嵌套子查询，返回部门名称为“SALES”的部门编号30；然后WHERE之后的子查询，查询部门编号为30的所有员工编号，查询的结果可能是单列多行；最后在外部的SELECT语句中使用IN操作符查询出部门编号为30的所有员工信息，即就职于“SALES”部门工作的所有员工信息。



注意

在实际应用中，应该尽量不要使用嵌套子查询技术，因为嵌套的层次越多，其查询性能越低。推荐使用表连接的方式。

9.7.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1179-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16727-1-1.html

9.8 嵌套、连接和简单查询分别适用于什么情况

9.8.1 问题描述

刚接触 Oracle 数据库，对知识掌握的比较模糊。今天我编写了一条 SQL 语句，发现使用嵌套查询和连接查询都可以获取到我想要的数据，并且查询出的结果数据是一模一样的。我想问一下：嵌套查询、连接查询和简单查询分别适用于什么情况？它们之间有什么区别？

9.8.2 解决方法

这个就要具体问题具体分析了，以我的观点来看，这个必须要自己在实际的 SQL 练习或者项目中去体会，没有固定要用什么方式。初学者实现就行，但是数据库管理员要做的更多的是考虑效率问题。

总的来说查询都是简单为好，复杂的嵌套查询会影响效率，基本就是用“SELECT * FROM table_name WHERE 条件”这样的简单查询就好。而嵌套查询和连接查询都需要视情况而定，比如我要写一个查询表 1 与表 2 中 id 对应，并且表 2 其中一个字段分数 score 值大于 60 的记录，可以使用嵌套查询：

```
SELECT * FROM table1
WHERE id IN (SELECT id FROM table2 WHERE score>60);
```

同样我们还可以使用简单的连接查询：

```
SELECT * FROM table1 a,table2 b
WHERE a.id=b.id AND b.score>60;
```

9.8.3 知识扩展——使用等号实现简单连接查询

连接查询就是借助两个表之间的关联关系进行数据查询，它是关系数据库查询最主要的特征。简单连接查询需要使用逗号将两个或多个表进行分隔，并通过 WHERE 子句建立表与表之间的关系，即通过使用相等比较符（=）指定连接条件的连接查询，这种连接查询主要用于检索主从表之间的相关数据，语法格式如下：

```
SEELCT table1.column1 [, table1.column2 ,[, ...]],table2.column1 [,table2.
column2 [ , ...]]
FROM table1,table2
WHERE table1.column1=table2.column2;
```

例如显示学生的信息及考试成绩，学生信息和考试成绩分别记录在不同的表中（student 表和 score 表），它们之间存在主从关系，即考试成绩表中的 stuid 字段引用学生信息表中的 id 字段。使用简单的连接查询即可创建它们的关系并显示出学生信息及考试成绩，如下：

```
SQL> SELECT * FROM student;
ID          NAME              AGE            SEX
----------  ----------------  -------------  -----------
1           马向林              22              女
2           殷国鹏              22              男
3           王丽丽              22              女
4           马林立              23              女
5           张小强              25              男

SQL> SELECT * FROM score;

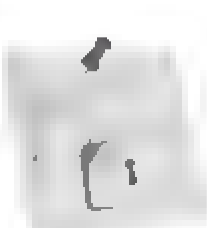
ID          RESULT     STUID
----------- ---------- ----------
1           89         1
2           90         2
3           84         3
4           78         4

SQL> SELECT student.id,student.name,student.age,student.sex,score.result
  2  FROM student,score
  3  WHERE student.id=score.stuid;

ID          NAME        AGE        SEX          RESULT
----------  ---------  ---------  -----------  --------------
1           马向林        22          女              89
2           殷国鹏        22          男              90
3           王丽丽        22          女              84
4           马林立        23          女              78
```

在 student 表中有 5 条数据，学号分别为 1、2、3、4、5，在 score 表中有 4 条数据，分别引用 student 表中学号为 1、2、3、4 的数据，故证明学号为 5 的学生没有参加此次考试。在连接查询后，只显示了参加考试的 4 条学生信息和成绩。

9.8.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1154-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-17035-1-1.html

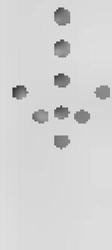
9.9 使用内连接查询问题

9.9.1 问题描述

这是一道考试的问题。有两个表：学生表包含学号、姓名两列；选课表包含学号、课程

两列，要求查询所有选高数的学生的姓名。我当时写的 SQL 语句如下：

```
SELECT 学生表.姓名 FROM 学生表
INNER JOIN 选课表
ON 学生表.学号 = 选课表.学号
WHERE 选课表.课程 = '高数';
```



这样写对吗？如果不对，应该如何写？请给我正确的答案，并简单说明。

9.9.2 解决方法

没有错，你写的是正确的。“学生表 INNER JOIN 选课表 ON 学生表.学号 = 选课表.学号”相当于“学生表 , 选课表 WHERE 学生表.学号 = 选课表.学号”，其作用就是将两个表按照学号相等连接成一个表，该表的结构如下：

```
学生表.学号  学生表.姓名  选课表.学号  选课表.课程
```

至于后面的“WHERE 选课表.课程='高数'”就是从连接之后的表中选择符合要求的数据，所以你写的完全正确。

9.9.3 知识扩展——使用 INNER JOIN 实现内连接

在连接查询时，多个表之间可以使用英文逗号进行分隔。除了这种形式以外，SQL 还支持使用 INNER JOIN 关键字进行内连接。内连接用于返回满足连接条件的记录，语法如下：

```
SELECT table1.column,table2.column FROM table1
[INNER] JOIN table2
On table1.column1=table2.column2;
```

其中：

- **INNER JOIN** 表示内连接。
- **ON** 用于指定连接条件。

下面使用内连接的 SQL 语句查询参加考试的所有学生姓名和分数，如下：

```
SQL> SELECT stu.id AS "学号",stu.name AS "姓名",se.result AS "分数"
  2  FROM student stu INNER JOIN score se
  3  ON stu.id=se.stuid;

学号                  姓名                分数
----------------    ----------------    ----------------
1                   马向林              9
2                   殷国鹏              90
3                   王丽丽              84
4                   马林立              78
```



内连接是最常用的连接查询方式，使用 INNER JOIN 关键字进行指定。如果只使用 JOIN 关键字，默认表示内连接。

9.9.4 触类旁通

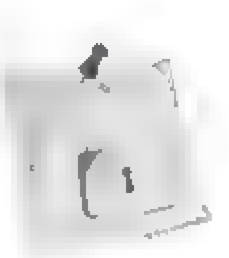


SQL 中 INNER JOIN 与 WHERE 有区别吗？
网络课堂：http://bbs.itzcn.com/thread-16731-1-1.html

在 SQL 中，INNER JOIN 与 WHERE 有区别吗？它们之间可以互相代替吗？如果不可以，哪种情况下不可以呢？

单从 INNSER JOIN 连接与 WHERE 子句来看，它们的区别不大，可以达到相同的效果。前者是显示连接，后者是隐式连接。不过，使用 INNER JOIN 的显示连接方式更适合数据库新的语言规范，已经逐渐替代使用 WHERE 子句的隐式连接方式。

9.9.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1155-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1156-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-16730-1-1.html

9.10 SQL 查询语句中的 LEFT OUTER JOIN 含义

9.10.1 问题描述

刚接触 SQL 语句，看到网上的一个例子中多处使用到了 LEFT OUTER JOIN 语句，这条语句是什么意思？在什么情况下需要使用该语句进行查询？它有什么作用？

9.10.2 解决方法

LEFT OUTER JOIN 表示的是左外连接，也被称为左连接。简单来说，如果 SQL 语句使用左连接，则保留在 LEFT OUTER JOIN 之前的表中的所有行，即左表中的所有行，哪怕在右表（LEFT OUTER JOIN 之后紧跟的表）中没有匹配的行，左表中的行也将全部显示。同理，若是 RIGHT OUTER JOIN，则保留右表中的所有行，即使在左表中没有匹配的行，右表中的行也将全部显示出来。

9.10.3 知识扩展——使用 OUTER JOIN 实现外连接

内连接时，返回查询结果集合中的仅是符号查询条件（WHERE 搜索条件或 HAVING 条件）和连接条件的行。而采用外连接时，它返回到查询结果集合中的不仅包含符合连接条件

的行，而且还包括左表（左外连接时）、右表（右外连接时）或两个连接表（全外连接时）中的所有数据行。外连接语法如下：

```
SELECT table1.column,table2.column FROM table1
[LEFT|RIGHT] OUTER JOIN table2
ON table1.column1=table2.column2;
```

对于外连接，Oracle 中可以使用加号（+）来表示，也可以使用 LEFT、RIGHT 和 FULL OUTER JOIN 关键字。

外连接可以分为下面这 3 类：

- ❑ 左外连接（LEFT OUTER JOIN 或 LEFT JOIN）。
- ❑ 右外连接（RIGHT OUTER JOIN 或 RIGHT JOIN）。
- ❑ 全外连接（FULL OUTER JOIN 或 FULL JOIN）。

使用外连接，列出与连接条件相匹配的行，并且列出左表（左外连接时）、右表（右外连接时）或两个表（全外连接时）中，所有符合检索条件的数据行。

1. 左外连接

左外连接也称为左连接，其结果集既包含连接表中的匹配行数据，也包括左连接表中所有满足检索条件的行。左连接表是指连接操作语句中 LEFT OUTER JOIN 操作符左边的连接表。

下面使用左连接查询本次考生的基本信息，如下：

```
SQL>  SELECT student.id 学号,name 姓名,age 年龄,sex 性别,result 分数
  2  FROM student
  3  LEFT OUTER JOIN
  4  score
  5  ON score.stuid=student.id;
学号        姓名       年龄                 性别           分数
---------- -------- ---------------- ----------- --------------
1          马向林      22                   女             89
2          殷国鹏      22                   男             90
3          王丽丽      22                   女             84
4          马林立      23                   女             78
5          张小强      25                   男
6          白雪        22                   女
已选择 6 行。
```

从查询结果可以看出，结果集中也包括那些在成绩表（score 表）中无记录的考生，即没有参加考试的学生，分数一列（result）显示为空值。

如果使用加号（+）建立左连接，那么上述左外连接语句等价于如下语句：

```
SQL>  SELECT student.id 学号,name 姓名,age 年龄,sex 性别,result 分数
  2  FROM student,score
  3  WHERE score.stuid(+)=student.id;
```

2. 右外连接

右外连接也称右连接，其结果集既包括连接表的匹配行数据，也包括右连接表中所有满

足检索条件的行。右连接表是连接操作语句中 RIGHT OUTER JOIN 操作符右边的连接表。

使用右外连接，检索所有参加考试的学生信息，如下：

```
SQL>  SELECT student.id 学号,name 姓名,age 年龄,sex 性别,result 分数
  2  FROM student
  3  RIGHT OUTER JOIN
  4   score
  5   ON score.stuid=student.id;

学号          姓名          年龄          性别          分数
----------  -------- --  ----------  ------ ----- -  ------------
1             马向林         22           女              89
2             殷国鹏         22           男              90
3             王丽丽         22           女              84
4             马林立         23           女              78
```

从查询结果可以看出，其结果集中包含了考生成绩表（score）中的所有数据，即参加本次考试的所有学生分数及该学生的基本信息。

如果使用加号（+）实现右连接，上述语句等价于下面的语句：

```
SQL>  SELECT student.id 学号,name 姓名,age 年龄,sex 性别,result 分数
  2  FROM student,score
  3  WHERE score.stuid=student.id(+);
```

3. 全外连接

全外连接是在结果中不仅包含符合连接条件的匹配行数据，而且包括两个连接表中的所有记录。

使用全外连接，检索学生信息表（student）与考生成绩表（score）中所包含的学号及对应的学生信息，如下：

```
SQL>  SELECT student.id 学号,name 姓名,age 年龄,sex 性别,result 分数
  2   FROM student
  3   FULL OUTER JOIN
  4   score
  5   ON score.stuid=student.id;

学号         姓名          年龄          性别          分数
----------  ----------  ----------  ----------  ----------
1            马向林         22           女             89
2            殷国鹏         22           男             90
3            王丽丽         22           女             84
4            马林立         23           女             78
5            张小强         25           男
6            白雪           22           女

已选择 6 行。
```

9.10.4 触类旁通



在 SQL 语句中使用 LEFT OUTER JOIN 的问题。

网络课堂：http://bbs.itzcn.com/thread-16732-1-1.html

我想问一下，在 SQL 语句中使用 LEFT OUTER JOIN 时，可以写成下面这样吗？

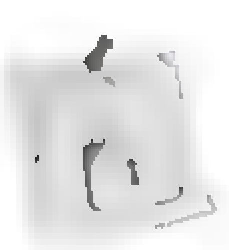
```
a LEFT OUTER JOIN b
WHERE b.column='BB'
```

这样写对吗？a 表与 b 表是主从关系，b 中的一个字段引用 a 表中的主键字段，使用 LEFT OUTER JOIN 之后可以不写关系表达式吗？

这样是不行的，LEFT OUTER JOIN 只是表明 SQL 语句中两个表之间的查询关系，即连接方式，并没有指明它们之间是通过什么来建立连接的，因此关系表达式是不可少的。除了需要指明关系表达式之外，还需要指明附加的条件“b.column='BB'”，SQL 语句如下：

```
a LEFT OUTER JOIN b
ON a.column = b.column
WHERE b.column= 'BB'
```

9.10.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1158-spid-35.html

视频教学：http://school.itzcn.com/video-vid-1159-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16729-1-1.html

9.11 什么是交叉连接

9.11.1 问题描述

在 SQL 语句中，有简单连接查询、内连接查询和外连接查询等连接方式，今天我又看到一个新的连接方式——交叉连接。什么是交叉连接？交叉连接返回多少条记录？

9.11.2 解决方法

从字面意义上来看，交叉连接可以实现两个表的交叉连接，所返回的结果将是这两个表中各行数据的所有组合，即 m*n 条记录。其中，m 指的是表 1 中的记录行数量，m 指的是表 2 中的记录行数量，例如 SCOTT 模式下的 emp 表中有 14 条记录、dept 表中有 4 条记录，下面将这两个表使用交叉连接的方式进行查询，如下：

```
SQL> SELECT empno 员工号,ename 员工姓名,sal 工资,dname 部门名称
  2  FROM emp
  3  CROSS JOIN dept;

    员工号 员工姓名          工资 部门名称
---------- ---------- ---------- --------------
      7369 SMITH             800 ACCOUNTING
      7499 ALLEN            1600 ACCOUNTING
      7521 WARD             1250 ACCOUNTING
      7566 JONES            2975 ACCOUNTING
      7654 MARTIN           1250 ACCOUNTING
      7698 BLAKE            2850 ACCOUNTING
...
      7900 JAMES             950 OPERATIONS
      7902 FORD             3000 OPERATIONS
      7934 MILLER           1300 OPERATIONS

已选择 56 行。
```

9.11.3 知识扩展——使用 CROSS JOIN 实现交叉连接

使用 CROSS JOIN 关键字，可以实现两个表的交叉连接，所得到的结果将是这两个表中各行数据的所有组合，即这两个表所有数据行的笛卡尔积。

交叉连接与简单连接操作非常相似，不同的是，使用交叉连接时，在 FROM 子句中多个表名之间不是用逗号，而是使用 CROSS JOIN 关键字隔开。另外，在交叉连接中不需要使用关键字 ON 限定连接条件，但是可以添加 WHERE 子句设置连接条件。

下面使用交叉连接，查询本次参加考试的马向林同学的成绩及基本信息，如下：

```
SQL>  SELECT student.id 学号,name 姓名,age 年龄,sex 性别,result 分数
  2  FROM student
  3  CROSS JOIN
  4  score
  5  WHERE score.stuid=student.id AND name='马向林';

学号          姓名          年龄--          性别              分数
---------- ---------------------------------------------------------
1           马向林          22              女                89
```

9.11.4 触类旁通



如何将两条 SQL 语句的查询结果建立交叉连接？

网络课堂：http://bbs.itzcn.com/thread-16734-1-1.html

统计 SCOTT 模式下的 dept 表中的记录数量：

```
SQL> select COUNT(*) FROM dept;
  COUNT(*)
----------
         4
```

统计 SCOTT 模式下的 emp 表中员工工资大于 1500 的记录数量：

```
SQL> SELECT COUNT(*) FROM emp WHERE sal>1500;
  COUNT(*)
----------
         7
```

现在我想要把这两条 SQL 语句统计出来的总数相乘，成为 CROSS JOIN 连接的返回结果总数，并查询出工资大于 1500 的员工号、姓名、工资和所在部门名称。应如何编写 SQL 语句？

你已经获取了 dept 表中的总数量为 4，工资大于 1500 元的员工总数为 7，也就表明我们使用 CROSS JOIN 将这两种查询结果建立交叉连接之后，返回的结果总数应该为 28。这道题并不难，可以把每条 SQL 语句查询的结果当做是一个表，也就是说，现在有两个表，一个表如下：

```
SQL> SELECT * FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

另一个表如下：

```
SQL> SELECT * FROM emp;

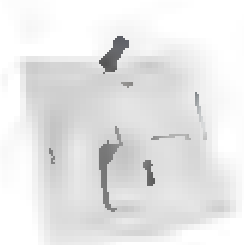
EMPNO   ENAME    JOB        MGR    HIREDATE     SAL    COM     DEPTNO
------  -------- ---------  ------ -----------  -----------   ------------
7369    SMITH    CLERK      7902   17-12月-80   800             20
7499    ALLEN    SALESMAN   7698   20-2月 -81   1600            30
...
7934    MILLER   CLERK      7782   23-1月 -82   1300            10
已选择14行。
```

将这两个表使用 CROSS JOIN 连接查询，如下：

```
SQL> SELECT empno 员工号,ename 员工姓名,sal 工资,dname 部门名称
  2  FROM (
  3  SELECT * FROM dept)
  4  CROSS JOIN
  5  (SELECT * FROM emp WHERE sal>1500);
```

```
员工号        员工姓名        工资        部门名称
---------- ---------- ---------- --------------
7499       ALLEN            1600   ACCOUNTING
7566       JONES            2975   ACCOUNTING
...
7839       KING             5000   OPERATIONS
7902       FORD             3000   OPERATIONS
已选择 28 行。
```

9.11.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1160-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16733-1-1.html

9.12 Oracle 中关于 UNION 排序的问题

9.12.1 问题描述

先看一段代码，如下：

```
SQL> COLUMN a dummy NOPRINT
SQL> SELECT 'sing' AS "My dream", 3 a dummy
  2  FROM dual
  3  UNION
  4  SELECT 'I''d like to teach', 1
  5  FROM dual
  6  UNION
  7  SELECT 'the world to', 2
  8  FROM dual
  9  ORDER BY 2;
```

执行结果如下：

```
My dream
-----------------
I'd like to teach
the world to
sing
```

结果为什么会是这样？还有 COLUMN a dummy NOPRINT 是什么意思？3、2、1 又是何意？“ORDER BY 2”中的 2 指的是第 2 列吗？“3 a dummy”是指对 3 进行更名吗？为什么

后面的2、1不需要添加a_dummy呢？刚学Oracle，很多不懂，请不要见笑！

9.12.2 解决方法

COLUMN a_dummy NOPRINT表示a_dummy这一列不输出显示，NOPRINT表示的意思就是"NOT PRINT"（不显示）。这样就只显示一列数据"My dream"，而3、2、1是一个整数列a_dummy上的值，由于该SQL语句使用了"ORDER BY 2"，就表示需要按照第二列进行排序，因此最后的结果为：

```
My dream
-------------------
I'd like to teach
the world to
sing
```

由于是UNION连接，所以对3这列更名a_dummy就意味着对整个结果列进行更名了，而不需要对下面的2、1进行更名。

9.12.3 知识扩展——使用UNION获取两个结果集的并集

使用集合操作符可以将两个或者多个查询返回的行组合起来，表9-1给出了常用的集合操作符。

表9-1 集合操作符

操作符	说明
UNION [ALL]	将多个查询结果集合并，形成一个新的结果集。如果指定ALL，则包括重复的行；如果不指定ALL，则对重复的行只保留一行。相当于结果集的OR运算
INTERSECT	返回多个查询检索出来的公共行。相当于结果集的AND运算
MINUS	返回多个查询检索出来的结果集的差集


提示

本节将介绍UNION操作符，其他各个操作将在后面的各小节中进行说明。

使用UNION操作符的语法如下：

```
select_statement UNION [ ALL ] select_statement
    [UNION [ ALL ] select_statement ] [ … ]
```

其中，select_statement是查询的SELECT语句；ALL选项表示将所有行合并到结果集中，不指定该项，则只保留重复行中的一行。

下面使用UNION ALL操作符，获取工资大于2500的员工信息与部门编号为30的员工信息的并集，即工资大于2500或者所在部门编号为30的员工信息。如下：

```
SQL> SELECT empno 编号,ename 姓名,sal 工资,deptno 所在部门号
```

```
  2  FROM emp
  3  WHERE sal>2500
  4  UNION ALL
  5  SELECT empno 编号,ename 姓名,sal 工资,deptno 所在部门号
  6   FROM emp
  7   WHERE deptno=30;

      编号    姓名            工资        所在部门号
    ---------- ---------- ---------- ----------
      7566   JONES          2975          20
      7698   BLAKE          2850          30
      7788   SCOTT          3000          20
      7839   KING           5000          10
      7902   FORD           3000          20
      7499   ALLEN          1600          30
      7521   WARD           1250          30
      7654   MARTIN         1250          30
      7698   BLAKE          2850          30
      7844   TURNER         1500          30
      7900   JAMES           950          30

已选择 11 行。
```

执行上述语句，将两个 SELECT 语句的查询结果合并在一起，但是并没有消除重复的行，例如员工编号为 7698 的行。



在进行 UNION 运算时，应该保证每个查询语句的列表具有相同的列或列的表达式。

下面再次使用 UNION 操作符，但是不指定 ALL 关键字，获得工资大于 2500 或者所在部门编号为 30 的员工信息，如下：

```
SQL>  SELECT empno 编号,ename 姓名,sal 工资,deptno 所在部门
  2   FROM emp
  3   WHERE sal>2500
  4   UNION
  5   SELECT empno 编号,ename 姓名,sal 工资,deptno 所在部门
  6    FROM emp
  7    WHERE deptno=30;

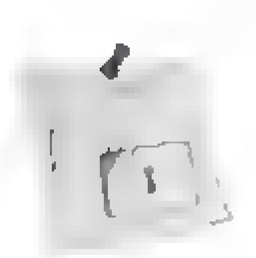
      编号    姓名            工资        所在部门号
    ---------- ---------- ---------- ----------
      7499   ALLEN          1600          30
      7521   WARD           1250          30
      7566   JONES          2975          20
      7654   MARTIN         1250          30
```

```
      7698    BLAKE          2850         30
      7788    SCOTT          3000         20
      7839    KING           5000         10
      7844    TURNER         1500         30
      7900    JAMES           950         30
      7902    FORD           3000         20

已选择 10 行。
```

比较两次的查询结果，不指定 ALL 的 UNION 操作符将获取到的是不重复的记录行。

9.12.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1180-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16735-1-1.html

9.13 Oracle 中 UNION 和 MINUS 的问题

9.13.1 问题描述

table1 表中有一个字段 num，并记录有 1、2、3、4、5 五个值；table2 表中也同样的含有一个字段，并记录有 4、5、6、7、8 八个值。使用如下的 SQL 语句对两个表进行连接查询：

```
SELECT * FROM table1 MINUS SELECT * FROM table2;
```

获取的结果应该为 1、2、3、6、7、8 对吗？如果将上述语句中的 MINUS 改为 UNION，获取到的应该是 1、2、3、4、5、6、7、8 对吧？也就是说 MINUS 是 table1 表与 table2 表不同记录的集合，而 UNION 获取的是 table1 表与 table2 表的并集，是这样吗？

9.13.2 解决方法

不是这样的，你的理解有误。下面我以真实的例子来给你讲述正确的理解。

这是 table1 中的记录：

```
SQL> SELECT * FROM table1;


NUM
----------
1
2
3
```

```
4
5
```

这是 table2 中的记录：

```
SQL> SELECT * FROM table2;

NUM
----------
4
5
6
7
8
```

使用 MINUS 对 table1 表和 table2 表进行连接查询，如下：

```
SQL> SELECT * FROM table1 MINUS SELECT * FROM table2;
NUM
----------
1
2
3
```

使用 UNION 对 table1 表和 table2 表进行连接查询，如下：

```
SQL>  SELECT * FROM table1 UNION SELECT * FROM table2;

NUM
----------
1
2
3
4
5
6
7
8
已选择 8 行。
```

从查询的结果来看，你所理解的 UNION 可以认为是正确的，是两个“集合”的并集，而 MINUS 并不是两个“集合”的不同元素，而是差集。

9.13.3 知识扩展——使用 MINUS 获取两个结果集的差集

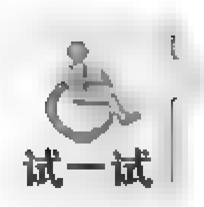
SQL 语言中的 MINUS 集合运算，表示获得给定集合之间的差异，也就意味着所得到的结果集中，其中的元素仅存在于前一个集合中，而不存在于另一个集合。

下面使用 NINUS 操作符，获得员工工资大于2500，但所在部门编号不为 30 的员工信息，如下：

使用 MINUS 操作符，获得员工编号大于 7800 但是所在部门编号不是 10 的员工信息。如下：

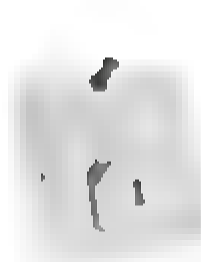
```
SQL>  SELECT empno 编号,ename 姓名,sal 工资,deptno 所在部门号
  2    FROM emp
  3    WHERE sal>2500
  4   MINUS
  5   SELECT empno 编号,ename 姓名,sal 工资,deptno 所在部门号
  6   FROM emp
  7   WHERE deptno=30;

      编号    姓名            工资          所在部门号
---------- ---------- ---------- ----------
      7566   JONES           2975          20
      7788   SCOTT           3000          20
      7839   KING            5000          10
      7902   FORD            3000          20
```



结合前面介绍的集合运算符，在一次执行语句中，可以根据需要对这些运算符进行混合使用。默认情况下，执行顺序自左至右，但是可以使用括号改变这个执行顺序。

9.13.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1182-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16736-1-1.html

第 10 章　PL/SQL 基础

PL/SQL 是一种过程化 SQL 语言，是 Oracle 数据库对 SQL 语言的扩展。使用 PL/SQL 可以编写具有很多高级功能的程序，当通过多条 SQL 语句实现功能时，每条 SQL 语句都需要在客户端和服务器端传递，因此占用了大量的网络宽带，消耗了大量的时间。而使用 PL/SQL 程序则是由于程序代码存储在数据库中，用户只需要在客户端发出调用 PL/SQL 的执行命令即可，在整个过程中网络只传输了很少的数据，从而缩短了网络传输的占用时间，提高了程序的执行性能。

在本章中，主要介绍 PL/SQL 语言的编写规范、程序块的结构、常量和变量的声明以及常用的数据类型、运算符和表达式、控制流程语句的使用、复合变量的用法以及游标的使用，最后还将进一步介绍异常处理。

10.1　PL/SQL 和 SQL 语言的不同之处

10.1.1　问题描述

SQL 语言是高级的非过程化编程语言，是沟通数据库服务器和客户端的重要工具。在数据库中，SQL 语句可以嵌套，因此具有极大的灵活性和强大的功能。那么 PL/SQL 和 SQL 是否一样呢，如果不一样，和 SQL 有什么不同之处？

10.1.2　解决方法

PL/SQL 是 Oracle 对标准数据库语言的扩展，Oracle 公司已经将 PL/SQL 整合到 Oracle 服务器和其他工具中了，近几年中更多的开发人员和 DBA 开始使用 PL/SQL。PL/SQL 不是一个独立的产品，而是一个整合到 Oracle 服务器和 Oracle 工具中的技术，可以把 PL/SQL 看作 Oracle 服务器内的一个引擎，SQL 语句执行者处理单个的 SQL 语句，PL/SQL 引擎处理 PL/SQL 程序块。

PL/SQL 的优点如下：

（1）PL/SQL 是一种高性能的基于事务处理的语言，能运行在任何 Oracle 环境中，支持所有数据处理命令。

（2）PL/SQL 支持所有 SQL 数据类型和所有 SQL 函数，同时支持所有 Oracle 对象类型。

（3）PL/SQL 可以被命名和存储在 Oracle 服务器中，同时也能被其他的 PL/SQL 程序或 SQL 命令调用，任何客户/服务器工具都能访问 PL/SQL 程序，具有很好的可重用性。

（4）PL/SQL 具有授权或撤销数据库其他用户访问 PL/SQL 程序的能力。

（5）PL/SQL 代码可以使用任何文本编辑器编写。

对于SQL，Oracle必须在同一时间处理每一条SQL语句，在网络环境下这就意味着每一个独立的调用都必须被Oracle服务器处理，需要占用大量的服务器时间，导致网络拥挤。而PL/SQL是以整个语句块发给服务器，这点就可以降低网络拥挤。

10.1.3 知识扩展——PL/SQL语言基础

PL/SQL，即Procedural Language/Structured Query Language的简称，是一种过程式语言，也是Oracle的专用语言。它是对标准SQL语言的扩展，全面支持SQL的数据操作、事务控制等。

SQL相对于PL/SQL来说，是一种声明式语言，没有流程控制、不存在变量，仅仅存在表或列，因此不能将某个SQL语句的执行结果传给另一个SQL语句，如果非要实现只有使用一条极其复杂的语句。而PL/SQL恰好可以弥补SQL语言的这些不足，在PL/SQL中可以通过定义变量来实现语句之间数据信息的传递，同时也可以使用控制流程语句IF和LOOP来控制程序的执行流程。

PL/SQL能够在运行Oracle的任何平台上运行，但不能像其他高级语言一样编译成可执行文件去执行。SQL*Plus是PL/SQL语言运行的基本工具，当程序第一句以DECLARE或BEGIN开头时，系统会自动识别出是PL/SQL语句，而不是直接的SQL命令。PL/SQL在SQL*Plus中运行时，当遇到斜杠（/）时才提交数据库执行，而不像SQL命令，遇到分号（;）就执行。

为了编写正确、高效的PL/SQL块，PL/SQL应用开发人员必须遵从特定的PL/SQL代码编写规则，否则会导致编译错误或运行错误。在编写PL/SQL代码时，应该遵从以下规则：

1．标识符命名规则

当在PL/SQL中使用标识符定义变量、常量时，标识符名称必须以字符开始，并且长度不能超过30个字符。另外，为了提高程序的可读性，Oracle建议用户按照以下规则定义各种标识符：

- ❑ 当定义变量时，建议使用v_作为前缀，例如v_sal、v_job等。
- ❑ 当定义常量时，建议使用c_作为前缀，例如c_rate。
- ❑ 当定义游标时，建议使用_cursor作为后缀，例如emp_cursor。
- ❑ 当定义异常时，建议使用e_作为前缀，例如e_integrity_error。
- ❑ 当定义PL/SQL表类型时，建议使用_table_type作为后缀，例如sal_table_type。
- ❑ 当定义PL/SQL表变量时，建议使用_table作为后缀，例如sal_table。
- ❑ 当定义PL/SQL记录类型时，建议使用_record_type作为后缀，例如emp_record_type。
- ❑ 当定义PL/SQL记录变量时，建议使用_record作为后缀，例如emp_record。

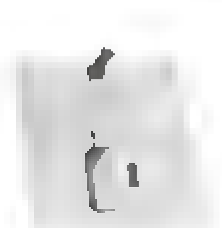
2．大小写规则

当在PL/SQL块中编写SQL语句和PL/SQL语句时，语句既可以使用大写格式，也可以使用小写格式。为了提高程序的可读性和性能，Oracle建议用户按照以下大小写规则编写代码：

- ❑ SQL关键字采用大写格式，例如SELECT，UPDATE，SET，WHERE等。
- ❑ PL/SQL关键字采用大写格式，例如DECLARE，BEGIN，END等。
- ❑ 数据类型采用大写格式，例如INT，VARCHAR2，DATE等。

- ❑ 标识符和参数采用小写格式，例如 v_sal、c_rate 等。
- ❑ 数据库对象和列采用小写格式，例如 emp、sal、ename 等。

10.1.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1188-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

10.2 Oracle 中 PL/SQL 程序块问题

10.2.1 问题描述

编写一个名称 showinfo 的程序块，并声明一些变量例如，编号 id、姓名 name、年龄 age 等。当编号 id 等于 0 时，则显示“对不起，用户不存在！”请问这道题怎么写，刚开始学习 Oracle 程序块，对很多知识点不是很懂，希望能够详细讲解，谢谢！

10.2.2 解决方法

首先需要了解 PL/SQL 程序块的结构，之后才能开始创建 showinfo 程序。

在 PL/SQL 程序中以 DECLARE 关键字声明的语句块中声明编号 id、姓名 name 以及年龄 age 的变量，其数据类型分别为 NUMBER、VARCHAR2(10)和 NUMBER。在以 BEGIN 关键字声明的语句块中对编号 id 进行判断。具体实现的代码如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  id NUMBER:=0;
  3  name VARCHAR2(10):='dcyandly';
  4  age NUMBER:=23;
  5  BEGIN
  6  IF (id=0) THEN
  7  DBMS_OUTPUT.PUT_LINE('对不起，用户不存在！');
  8  END IF;
  9  END;
 10  /
对不起，用户不存在！
PL/SQL 过程已成功完成。
```

10.2.3 知识扩展——PL/SQL 程序块

构成一个 PL/SQL 程序的基本结构是程序块。程序块由过程、函数和无名块 3 种形式组

成。这三种形式之间可以嵌套，并且在每一个程序块中都包含PL/SQL语句和SQL语句。典型的PL/SQL程序块结构的格式如下：

```
[ DECLARE declaration statements ; ]
BEGIN
    executable statements ;
[ EXCEPTION exception handling statements ; ]
END ;
/
```

上述格式中的语法参数说明如下：

❑ **DECLARE declaration_statements**

用于声明变量。PL/SQL程序块中需要使用的变量一般在DECLARE块中声明。

❑ **BEGIN … END**

PL/SQL程序块的主体部分。其中，还可以嵌套其他PL/SQL程序块。

❑ **executable_statements**

PL/SQL块中的可执行语句。

❑ **EXCEPTION exception_handling_statements**

用于处理PL/SQL程序块运行过程中可能出现的任何可执行错误。

❑ **/**

PL/SQL程序块需要使用正斜杠（/）结尾，才能被执行。

在Oracle中，PL/SQL程序块中的每一条语句都必须以分号结束，SQL语句可以是多行的，但分号表示该语句的结束。一行中可以有多条SQL语句，但是它们之间必须以分号分隔。PL/SQL程序的注释是由--表示。

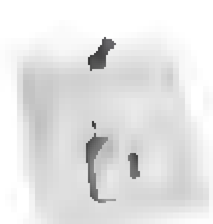
例如，使用PL/SQL语言计算一个乘法运算，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2      num1 NUMBER:=8;
  3      num2 NUMBER:=9;
  4      nums NUMBER;
  5  BEGIN
  6      nums:=num1*num2;
  7      DBMS_OUTPUT.PUT_LINE(nums);
  8  EXCEPTION
  9      WHEN OTHERS THEN
 10      DBMS_OUTPUT.PUT_LINE('出现异常');
 11  END;
 12  /
72
PL/SQL 过程已成功完成。
```

在上述程序中，为了在服务器端显示执行结果，需要使用SET SERVEROUTPUT ON命令。在DECLARE关键字表示的声明块中声明了数据类型均是NUMBER的3个变量分别是

num1、num2 和 nums，并为变量 num1 和 num2 赋予了初始值 8 和 9；接着使用 BEGIN 关键字标识可执行块的开始，在可执行块中包含了一条 PL/SQL 语句，该条语句是计算 num1*num2 的值，并赋值给变量 nums；然后使用 DBMS_OUTPUT.PUT_LINE(nums);语句来显示计算的结果；EXCEPTION 关键字表示异常处理块的开始。

10.2.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1189-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

10.3 请问如何利用 SQL 查询为 PL/SQL 变量赋值

10.3.1 问题描述

在 Oracle 中创建一个用户表 tab_user，并在 tab_user 表中添加一条数据，如下：

```
SQL> CREATE TABLE tab_user
  2  (
  3  userid NUMBER(4),
  4  username  CHAR(10),
  5  usersex CHAR(4),
  6  userage NUMBER(4)
  7  );
表已创建。
SQL> INSERT INTO tab_user VALUES(1,'one','女',23);
已创建 1 行。
```

请问怎样才能将使用 SQL 语句查询的用户信息赋值给 PL/SQL 变量呢？请各位指点一二，谢谢！

10.3.2 解决方法

首先需要在 DECLARE 关键字表示的声明块中声明数据类型为 NUMBER 和 VARCHAR2 的变量，接着使用 BEGIN 关键字标识可执行块的开始，在可执行块中使用 SELECT…INTO 语句分别为 NUMBER 和 VARCHAR2 类型的变量赋值，该值分别为 tab_user 表中 userid、username、usersex 以及 userage 列的值。最后调用 DBMS_OUTPUT.PUT_LINE 输出这些变量的值。具体的实现如下：

```
SQL>  SET SERVEROUTPUT ON
SQL>  DECLARE
```

```
  2     v_userid NUMBER(4);
  3     v_username VARCHAR2(10);
  4     v_usersex VARCHAR2(4);
  5     v_userage NUMBER(4);
  6   BEGIN
  7     SELECT userid,username,usersex,userage
  8     INTO v_userid,v_username,v_usersex,v_userage
  9     FROM tab_user;
 10     DBMS_OUTPUT.PUT_LINE('用户的编号为：' || v_userid) ;
 11     DBMS_OUTPUT.PUT_LINE('用户的名称为：' || v_username) ;
 12     DBMS_OUTPUT.PUT_LINE('性别为：' || v_usersex) ;
 13     DBMS_OUTPUT.PUT_LINE('年龄为：' || v_userage) ;
 14   END ;
 15   /
用户的编号为：1
用户的名称为：one
性别为：女
年龄为：23
PL/SQL 过程已成功完成。
```

10.3.3 知识扩展——变量和类型

在 PL/SQL 程序块中，经常会使用到变量和常量。常量用于声明一个不可更改的值，而变量则可以在程序中根据需要存储不同的值。在 PL/SQL 程序中常用的数据类型有 4 种，分别是：标量（Scalar）类型、复合（Composite）类型、参照（Reference）类型和 LOB（Large Object）类型。

在定义变量和常量时，其名称必须符合 Oracle 标识符的规定，如下：

- ❑ 变量名以字母开头，不区分大小写。
- ❑ 变量名由字母、数字以及$、#或_和特殊字符组成。
- ❑ 变量长度最多包含 30 个字符。
- ❑ 变量名中不能有空格。
- ❑ 尽可能避免缩写，用一些具有意义的单词命名。
- ❑ 不能用保留字命名。

1．变量

在 PL/SQL 程序中，最常用的变量是标量变量，当使用变量时需要声明变量，其语法如下：

```
variable_name data_type [ [ NOT NULL ] { := | DEFAULT } value ] ;
```

variable_name 表示定义变量的名称。NOT NULL 表示可以对变量定义非空约束。如果使用了此选项，则必须为该变量赋非空的初始值，并且不允许在程序其他部分将其值修改为 NULL。

例如，定义一个变量 num，并为其赋值如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     num NUMBER(4);
  3  BEGIN
  4     num:=500;
  5     DBMS_OUTPUT.PUT_LINE('获取变量的值为：' || num);
  6  END;
  7  /
获取变量的值为：500
PL/SQL 过程已成功完成。
```

2. 常量

声明常量时需要使用 CONSTANT 关键字，并且必须在声明时就为该常量赋值，而且在程序其他部分不能修改该常量的值。定义常量的语法格式如下：

```
constant_name CONSTANT data_type { := | DEFAULT } value ;
```

在上述语法格式中，constant_name 表示常量的名称；data_type 表示常量的数据类型；:=|DEFAULT 中，:=为赋值操作符。

例如，定义一个常量 num1，如下：

```
SQL>  SET SERVEROUTPUT ON
SQL>  DECLARE
  2     num1 CONSTANT NUMBER(4) := 123;
  3  BEGIN
  4     DBMS_OUTPUT.PUT_LINE('你输入的常量是：' || num1);
  5  END;
  7  /
你输入的常量是：123
PL/SQL 过程已成功完成。
```



PL/SQL 程序块中的赋值符号是冒号等号（:=），而不是常见的等号（=），并且在书写时不要将冒号与等号分开，也就是说两者之间不能存在空格。

3. PL/SQL 数据类型

对于常量与变量的数据类型，除了可以使用与 SQL 相同的数据类型以外，Oracle 还专门为 PL/SQL 程序块提供了表 11-1 所示的特定类型。

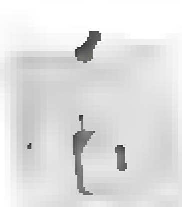
表 11-1　PL/SQL 数据类型

[illegible]	[illegible]
BOOLEAN	布尔型。取值为 TRUE、FALSE 或 NULL
BINARY_INTEGER	带符号整数，取值范围为 -2^{31}～2^{31}
NATURAL	BINARY_INTEGER 的子类型，表示非负整数
NATURALN	BINARY_INTEGER 的子类型，表示不为 NULL 的非负整数
POSITIVE	BINARY_INTEGER 的子类型，表示正整数

续表

类　型	说　明
POSITIVEN	BINARY_INTEGER 的子类型，表示不为 NULL 的正整数
SIGNTYPE	BINARY_INTEGER 的子类型，取值为-1、0 或 1
PLS_INTEGER	PLS_INTEGER 是专为 PL/SQL 程序使用的数据类型，它不可以在创建表的列中使用，PLS_INTEGER 数据类型表示一个有符号整数，表示的范围为-2^{31}～2^{31}。PLS_INTEGER 具有比 NUMBER 变量更小的表示范围，因此会占用更少的内存。PLS_INTEGER 能够更有效地利用 CPU，因此其运算可以比 NUMBER 和 BINARY_INTEGER 更快
SIMPLE_INTEGER	Oracle Database 11g 的新增类型。它是 BINARY_INTEGER 的子类型，其取值范围与 BINARY_INTEGER 相同，但不能存储 NULL 值。当使用 SIMPLE_INTEGER 值时，如果算法发生溢出，不会触发异常，只会简单地截断结果
STRING	与 VARCHAR2 相同
RECORD	一组其他类型的组合
REF CURSOR	指向一个行集的指针

10.3.4　网络课堂



视频教学：http://school.itzcn.com/video-vid-1191-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1190-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

10.4　PL/SQL 中 WHERE 后面的表达式怎么写

10.4.1　问题描述

有一个名称为 tab_user 的表，查询的结果如下：

```
SQL> SELECT * FROM tab_user;
    USERID  USERNAME   USER  USERAGE
----------- ---------- ---- ----------
         2  one         女        23
         1              女        23
```

在这里如何将表 tab_user 中 username 列为空的值提取并输出？WHERE 后面的表达式怎么写？请指点，谢谢！

10.4.2　解决方法

这里如果需要将 username 列为空的值提取出来需要使用到逻辑运算符 is null，具体的代码实现如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     v userid NUMBER(4);
  3     v username VARCHAR2(10);
  4     v_usersex VARCHAR2(4);
  5     v userage NUMBER(4);
  6   BEGIN
  7     SELECT userid,username,usersex,userage
  8     INTO v_userid,v_username,v_usersex,v_userage
  9     FROM tab user WHERE username is null;
 10     DBMS_OUTPUT.PUT_LINE('用户的编号为：' || v_userid) ;
 11     DBMS_OUTPUT.PUT_LINE('用户的名称为：' || v_username) ;
 12     DBMS OUTPUT.PUT LINE('性别为：' || v usersex) ;
 13     DBMS_OUTPUT.PUT_LINE('年龄为：' || v_userage) ;
 14  END ;
 15  /
用户的编号为：1
用户的名称为：
性别为：女
年龄为：23
PL/SQL 过程已成功完成。
```

这样就可以将 username 列为空的值提取出来了，是你想要的吧！

10.4.3 知识扩展——运算符与表达式

在 PL/SQL 程序中，当计算两个数据之间的差时，需要使用运算符-；当为常量和变量赋值时，需要使用运算符:=；当处理 PL/SQL 程序中的某个值是否为空时，需要使用逻辑运算符 is null 等；因此在 PL/SQL 程序中，为了满足 PL/SQL 程序各种处理的要求，PL/SQL 程序块允许在表达式中使用关系运算符与逻辑运算符。表 11-2、表 11-3 和表 11-4 将分别为大家列出了常用的一般运算符、关系运算符和逻辑运算符。

表 11-2 一般运算符

一般运算符	含义	一般运算符	含义
+	加号	:=	赋值号
-	减号	=>	关系号
*	乘号	..	范围运算符
/	除号	||	字符连接符

表 11-3 关系运算符

关系运算符	含义	关系运算符	含义
=	等于	>	大于
<>、!=、~=、^=	不等于	<=	小于或等于
<	小于	>=	大于或等于

表 11-4　逻辑运算符

逻辑运算符	含　义	逻辑运算符	含　义
is null	是空值	And	逻辑与
Between	介于两者之间	or	逻辑或
In	在一列值中间	Not	取反

例如，将 scott 模式下的 emp 表中字段 empno 的值为 7369 的一条信息输出，如下：

```
SQL>  SET SERVEROUTPUT ON
SQL>  DECLARE
  2     v_empno NUMBER(10);
  3     v_ename VARCHAR2(10);
  4     v_job   VARCHAR2(10);
  5     v_sal NUMBER(10);
  6   BEGIN
  7     SELECT empno,ename,job,sal INTO
  8     v_empno,v_ename,v_job,v_sal
  9     FROM scott.emp WHERE empno=7369;
 10     DBMS_OUTPUT.PUT_LINE('编号为：' || v_empno) ;
 11     DBMS_OUTPUT.PUT_LINE('名称为：' || v_ename);
 12     DBMS_OUTPUT.PUT_LINE('JOB 为：' || v_job);
 13     DBMS_OUTPUT.PUT_LINE('SAL 为：' || v_sal);
 14  END ;
 15  /
编号为：7369
名称为：SMITH
JOB 为：CLERK
SAL 为：800
PL/SQL 过程已成功完成。
```

10.4.4　触类旁通



如何获取一个表中不为空的数据？

网络课堂：http://bbs.itzcn.com/thread-662-1-1.html

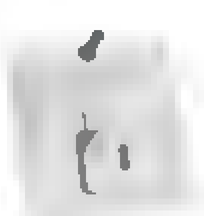
在一个名称为 tab_user 的表中有两条数据，其中一条数据的 username 列的值为空，那么怎样在 PL/SQL 程序中将不为空数据的信息输出呢？请路过的哥哥姐姐们指点一二，谢谢！

当需要提取 username 列为空的数据是使用逻辑运算符 is null，当提取 username 列不为空的数据就需要使用逻辑运算符 Not 了，来看一下实现的代码：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     v_userid NUMBER(4);
  3     v_username VARCHAR2(10);
  4     v_usersex VARCHAR2(4);
```

```
 5     v_userage NUMBER(4);
 6  BEGIN
 7     SELECT userid,username,usersex,userage
 8     INTO v_userid,v_username,v_usersex,v_userage
 9     FROM tab_user WHERE username is Not null;
10     DBMS_OUTPUT.PUT_LINE('用户的编号为：' || v_userid) ;
11     DBMS_OUTPUT.PUT_LINE('用户的名称为：' || v_username) ;
12     DBMS_OUTPUT.PUT_LINE('性别为：' || v_usersex) ;
13     DBMS_OUTPUT.PUT_LINE('年龄为：' || v_userage) ;
14  END ;
15  /
用户的编号为：2
用户的名称为：one
性别为：女
年龄为：23
PL/SQL 过程已成功完成。
```

10.4.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1192-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

10.5 关于 PL/SQL 中使用循环语句的问题

10.5.1 问题描述

有这样一个名称为 mydot 的表，在该表中有 id 一个字段，请问如何循环添加 3 条数据到 mydot 的表中，数据的编号从 1 开始。请各位哥哥姐姐帮帮忙，谢谢！

10.5.2 解决方法

你可以使用 WHILE 循环语句来实现，首先声明一个名称为 v_num 的变量，用来存储写入表 mydot 的 id。接着在 BEGIN 块中使用 WHILE 语句循环向 mydot 表中 id 列添加值。具体实现的代码如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  v_num NUMBER:=1;
  3  BEGIN
  4  WHILE v_num<=3
  5  LOOP
```

```
 6  INSERT INTO mydot VALUES(v num);
 7  v num:=v num+1;
 8  END LOOP;
 9  END;
10  /
PL/SQL 过程已成功完成。
SQL> SELECT * FROM mydot;
    ID
----------
    1
    2
    3
```

10.5.3 知识扩展——控制结构

在 Oracle 中，PL/SQL 程序能处理各种基本的控制结构，例如，条件结构、循环结构以及顺序结构等。在 PL/SQL 程序块中不仅可以嵌入 SQL 语句，还可以嵌入条件分支语句（例如 IF、CASE 等）、循环语句（例如，LOOP）以及顺序控制语句（例如、GOTO、NULL 等）。

1. 条件语句

在 Oracle 中主要提供了两种条件选择语句来对程序进行逻辑控制，这两种选择条件语句分别是：IF 条件语句和 CASE 表达式。

❑ IF 条件语句

IF 条件语句可以包含 IF、ELSIF、ELSE、THEN 以及 END IF 等关键字。其完整的语法形式如下：

```
IF condition1 THEN
    statements1
[ ELSIF condition2 THEN
    statements2 ] [ , … ]
[ ELSE
    statements3 ]
END IF ;
```

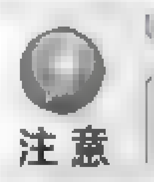
在上述语法格式中，condition1 和 condition2 是布尔表达式，其值为真或假；statements1、statements2 和 statements3 代表 PL/SQL 语句。含义是如果 condition1 为真，则执行 statements1；如果 condition1 为假而 condition2 为真，则执行 statements2；如果 condition1 和 condition2 都为假，则执行 statements3。

例如，使用 IF 条件语句判断年龄 18 所处的等级，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     age NUMBER(4):=18;
  3  BEGIN
  4     IF age>= 18 THEN
```

```
  5          DBMS OUTPUT.PUT LINE('已成年') ;
  6      ELSIF age< 18 THEN
  7          DBMS OUTPUT.PUT LINE('未成年') ;
  8      ELSE
  9          DBMS OUTPUT.PUT LINE('系统出错') ;
 10      END IF ;
 11  END ;
 12  /
已成年
PL/SQL 过程已成功完成。
```

在上述代码 DECLARE 中定义了一个变量 age，并为其赋值 18。接着在 BEGIN 中使用 IF 条件语句进行判断。如果 age 大于指定的值 18 时，则输出“已成年”，如果小于指定的值 18 则输出“未成年”，否则就输出“系统出错”。



注意

IF 语句是基本的选择结构语句。每一个 IF 语句都有 THEN，以 IF 开头的语句行不能跟语句结束符：分号（;），每一个 IF 语句以 END IF 结束；每一个 IF 语句有且只能有一个 ELSE 语句相对应。

❑ **CASE 表达式**

Oracle 中 CASE 分为两种类型分别是：简单 CASE 表达式和搜索 CASE 表达式。其中简单 CASE 表达式的作用是使用表达式来确定返回值，而搜索 CASE 表达式的作用是使用条件确定返回值。下面首先来看简单 CASE 表达式。



注意

在功能上，CASE 表达式和 IF 条件语句很相似，可以说是 CASE 表达式基本上可以实现 IF 条件语句能够实现的所有功能。从代码结构上来讲，CASE 表达式具有很好的阅读性。

（1）简单 CASE 表达式

简单 CASE 表达式使用嵌入式的表达式来确定返回值，其语法如下：

```
CASE search expression
    WHEN expression1 THEN result1 ;
    WHEN expression2 THEN result2 ;
     …
    WHEN expressionN THEN resultN ;
    [ ELSE default_result ; ]
END CASE ;
```

在上述语法格式中，search expression 表示待求值的表达式；expression1 表示要与 search expression 进行比较的表达式，如果二者的值相等，则返回 result1，否则进入下一次比较；default_result 表示如果所有的 WHEN 子句中的表达式的值都与 search_expression 不匹配，则返回 default result，即默认值；如果不设置此选项，而又没有找到匹配的表达式，则 Oracle 将报错。

例如，使用 CASE 表达式判断“已成年”对应的年龄段，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     age VARCHAR2(10) := '已成年' ;
  3  BEGIN
  4     CASE age
  5         WHEN '已成年' THEN DBMS_OUTPUT.PUT_LINE('大于等于 18 岁') ;
  6         WHEN '未成年' THEN DBMS_OUTPUT.PUT_LINE('小于 18 岁') ;
  7         WHEN '年龄不合格' THEN DBMS_OUTPUT.PUT_LINE('小于 0 岁') ;
  8         ELSE DBMS_OUTPUT.PUT_LINE('系统错误') ;
  9     END CASE ;
 10  END ;
 11  /
大于等于 18 岁
PL/SQL 过程已成功完成。
```

在上述代码 DECLARE 中定义了一个变量 age，并为其赋值“已成年”。接着在 BEGIN 中使用 CASE 表达式对 age 进行判断，最后结果进行输出。

（2）搜索 CASE 表达式

搜索 CASE 表达式使用条件来确定返回值，其语法如下：

```
CASE
    WHEN condition1 THEN result1 ;
    WHEN condition2 THEN result2 ;
     …
    WHEN conditionN THEN resultN ;
    [ ELSE default result ; ]
END CASE ;
```

与简单 CASE 表达式相比较，可以发现 CASE 关键字后面不再跟随待求表达式，而 WHEN 子句中的表达式也换成了条件语句（condition），其实搜索 CASE 表达式就是将待求表达式放在条件语句中进行范围比较，而不再像简单 CASE 表达式那样只能与单个的值进行比较。

例如，使用搜索 CASE 表达式实现年龄 20 所对应的年龄段，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     age NUMBER(4):=20;
  3  BEGIN
  4     CASE
  5         WHEN age >= 18 T HEN DBMS_OUTPUT.PUT_LINE('已成年') ;
  6         WHEN age < 18 THEN DBMS_OUTPUT.PUT_LINE('未成年') ;
  7     END CASE ;
  8  END ;
  9  /
已成年
PL/SQL 过程已成功完成。
```

2. 循环语句

循环语句一般由循环体和循环结束条件组成，循环体是指被重复执行的语句集，而循环结束条件则用于终止循环。如果没有循环结束条件，或循环结束条件永远返回 FALSE，则循环将陷入死循环。

在 PL/SQL 程序中，循环语句主要包括 LOOP 循环语句、WHILE 循环语句以及 FOR 循环语句 3 种。首先来看 LOOP 循环语句。

❑ LOOP 循环语句

LOOP 循环语句是最基本的循环语句，该循环语句以 LOOP 开始，以 END LOOP 结束，其语法如下：

```
LOOP
   statements
   EXIT [WHEN condition]
END LOOP;
```

在上述语法格式中，statements 是 LOOP 循环体中的语句块。无论是否满足条件，statements 至少会被执行一次。当 condition 为 TRUE 时，会退出循环，并执行 END LOOP 后的相应操作。

例如，使用 LOOP 循环语句输出数字从 1 到 3，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     num NUMBER(4):=1;
  3  BEGIN
  4     LOOP
  5         DBMS_OUTPUT.PUT_LINE(num) ;
  6         num:=num+1;
  7         EXIT WHEN num > 3 ;
  8     END LOOP ;
  9  END ;
 10  /
1
2
3
PL/SQL 过程已成功完成。
```

❑ WHILE 循环语句

WHILE 循环是在 LOOP 循环的基础上添加循环条件，也就是说只有满足 WHILE 条件后，才会执行循环体中的内容。其语法如下：

```
WHILE condition
LOOP
   statements ;
END LOOP ;
```

在上述语法格式中，当 condition 为 TRUE 时，PL/SQL 执行器会执行 statements；而当条

件为 FALSE 或 NULL 时，会退出循环，并执行 END LOOP 后的语句。

例如，使用 WHILE 循环语句输出 3 遍“欢迎你”，如下：

```
SQL>  SET SERVEROUTPUT ON
SQL>  DECLARE
  2     num NUMBER:=1;
  3    BEGIN
  4     WHILE num <= 3
  5     LOOP
  6         DBMS_OUTPUT.PUT_LINE('欢迎你') ;
  7         num:=num+1;
  8     END LOOP ;
  9   END ;
 10   /
欢迎你
欢迎你
欢迎你
PL/SQL 过程已成功完成。
```

- **FOR 循环语句**

FOR 循环是在 LOOP 循环的基础上添加循环次数，其语法形式如下：

```
FOR loop variable IN [ REVERSE ] lower bound .. upper bound
LOOP
    statements ;
END LOOP ;
```

在上述语法格式中，loop_variable 表示指定循环变量；REVERSE 指定在每一次循环中循环变量都会递减。这里循环变量首先被初始化为其终止值，然后在每一次循环中递减 1，直到达到其起始值；lower_bound 指定循环的起始值。在没有使用 REVERSE 的情况下，循环变量初始化为该起始值；upper_bound 指定循环的终止值，如果使用 REVERSE，循环变量就初始化为该终止值。

例如，使用 FOR 循环语句，输出数字 1 到 3，如下：

```
SQL>   SET SERVEROUTPUT ON
SQL>   DECLARE
  2     num NUMBER(4);
  3     BEGIN
  4     FOR num IN 1 .. 3
  5     LOOP
  6         DBMS_OUTPUT.PUT_LINE(num) ;
  7     END LOOP;
  8   END ;
  9   /
1
2
```

```
3
PL/SQL 过程已成功完成。
```

3. GOTO 和 NULL 结构

GOTO 和 NULL 语句不经常使用。

❑ **GOTO 结构**

GOTO 结构又称为跳转结构。在 PL/SQL 中使用 GOTO 可以使程序转到设定的标签，执行某个代码区域，实现逻辑分支结构。在 PL/SQL 中使用符号“<<>>”来创建标。例如：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     num INTEGER :=2;
  3  BEGIN
  4     IF num>1 THEN
  5         GOTO BIG LABLE;
  6     END IF;
  7  <<BIG LABLE>>
  8     DBMS_OUTPUT.PUT_LINE('跳到标签<<BIG_LABLE>>中');
  9  END;
 10  /
跳到标签<<BIG_LABLE>>中
PL/SQL 过程已成功完成。
```

在上述代码中，使用了 IF 条件语句进行判断，如果变量 num 的值大于 1，则使用 GOTO 结构跳转到<<BIG_LABLE>>中，进而执行该标签下面的语句。

❑ **NULL 结构**

NULL 结构，又称空操作或空值结构，在 PL/SQL 中是一类特殊的结构。在 PL/SQL 程序中使用 NULL 结构，表示什么操作也不做，仅仅起到占位符的作用。例如：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     num INTEGER :=2;
  3  BEGIN
  4     IF num>1 THEN
  5         NULL;
  6     ELSE
  7         DBMS OUTPUT.PUT LINE('num 的值比 1 大');
  8     END IF;
  9   END;
 10   /
PL/SQL 过程已成功完成。
```

在上述代码中，使用了 IF 条件语句进行判断，如果变量 num 的值大于 1，则执行 NULL 结构。

10.5.4 触类旁通



使用循环输出九九乘法表。

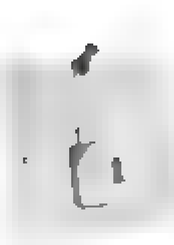
网络课堂：http://bbs.itzcn.com/thread-662-1-1.html

PL/SQL 程序中，如何使用循环将九九乘法表有规律的输出呢？请各位帮帮忙，谢谢！

这里我们使用 FOR 循环将乘法表输出，具体实现的代码如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  m NUMBER;
  3  BEGIN
  4  FOR i IN 1..9 LOOP
  5    FOR j IN 1..i LOOP
  6        m := i*j;
  7        DBMS_OUTPUT.PUT('*');
  8        DBMS_OUTPUT.PUT (i||'*'||j||'='||m||' ');
  9     END LOOP;
 10       DBMS OUTPUT.PUT LINE(NULL);
 11  END LOOP;
 12  END;
 13  /
1*1=1
2*1=2 2*2=4
3*1=3 3*2=6 3*3=9
4*1=4 4*2=8 4*3=12 4*4=16
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
PL/SQL 过程已成功完成。
```

10.5.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1193-spid-35.html

视频教学：http://school.itzcn.com/video-vid-1194-spid-35.html

视频教学：http://school.itzcn.com/video-vid-1195-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

10.6 PL/SQL 中如何获取某个变量的类型

10.6.1 问题描述

请问在 PL/SQL 程序中使用什么语句可以获得某个变量的类型呢？请各位帮帮忙，谢谢！

10.6.2 解决方法

在 PL/SQL 中一般使用%TYPE 类型的变量获取已知的变量类型，使用%ROWTYPE 类型的变量获取行类型。如下

```
DECLARE
  v_name emp.ename%TYPE;
BEGIN
  SELECT ename INTO v_name FROM emp
    WHERE empno=7788;
  DBMS_OUTPUT.PUT_LINE(v_name);
END;
```

上述代码中就是获取 scott.emp 表中 ename 列的数据类型 VARCHAR2(10)，然后检索一行一列数据，并将该数据的值赋给使用%TYPE 关键字声明的类型变量 v_name。

```
DECLARE
  v_dept dept%ROWTYPE;
BEGIN
  SELECT * into v_dept FROM dept
  WHERE deptno=30;
  DBMS_OUTPUT.PUT_LINE(v_dept.deptno);
  DBMS_OUTPUT.PUT_LINE(v_dept.dname);
  DBMS_OUTPUT.PUT_LINE(v_dept.loc);
END;
```

上述代码中就是定义了一个%ROWTYPE 类型的变量 v_dept，该变量的结构与 scott.dept 表的结构完全相同。并将该表中字段 deptno 为 30 的一行数据赋值给变量 v_dept。在输出时，使用变量 v_dept 点该行的某个列的值即可。

10.6.3 知识扩展——复合变量

复合变量与标量变量相对应，相对于标量变量，它可以将不同数据类型的多个值存储在一个单元中。当定义复合变量时，必须使用 PL/SQL 复合数据类型，常用的复合数据类型主要有 3 种，分别是：%TYPE 类型、自定义记录类型以及%ROWTYPE 类型。

1. %TYPE 类型

在声明变量时，不仅可以使用 Oracle 规定的数据类型，还可以使用%TYPE 关键字定义变量类型。使用%TYPE 关键字声明的变量类型与数据表中字段的数据类型相同，如下：

```
DECLARE
var name emp.ename%type;
```

在上述代码中，如果 emp 表中 ename 列的数据类型为 VARCHAR2(10)，那么变量 var_name 的数据类型就为 VARCHAR2(10)。

例如，使用%TYPE 关键字声明变量类型 v_num 和 v_age，从数据库中检索数据。如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     v num tab user.userid%TYPE;
  3     v age tab user.userage%TYPE;
  4  BEGIN
  5     SELECT userid,userage
  6     INTO v_num,v_age
  7     FROM tab user WHERE userid=1;
  8     DBMS_OUTPUT.PUT_LINE('用户的编号为：' ||v_num) ;
  9     DBMS_OUTPUT.PUT_LINE('年龄为：' ||v_age) ;
 10  END;
 11  /
用户的编号为：1
年龄为：23
PL/SQL 过程已成功完成。
```

2. 自定义记录类型

自定义记录类型是表示在记录类型的复合变量中可以存储多个标量值。当使用记录类型的变量时，首先需要定义记录的结构，然后才可以声明记录类型的变量。定义记录数据类型时必须使用 TYPE 语句，在该语句中指出将在记录中包含的字段以及数据类型。使用 TYPE 语句定义记录数据类型的语法格式如下：

```
TYPE record name is record(
field1 name  data type  [not null] [:=default value],
……
fieldn_name data_type [not null] [:=default_value]);
```

在上述语法格式中，record_name 表示自定义的记录数据类型名称，例如 NUMBER；field1_name 表示记录数据类型中的字段名；data_type 为该字段的数据类型。

例如，定义一个名称为 my_type 的记录类型，该记录类型由整数型的 v_num 和整数型的 v_age 基本类型变量组成。其中 my 为该类型的变量，引用记录类型变量的方法是“记录变量名.字段名”。如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
```

```
  2     TYPE my type is record
  3     (
  4     v num NUMBER,
  5     v age NUMBER
  6     );
  7     my my_type;
  8   BEGIN
  9     SELECT userid,userage
 10     INTO my
 11     FROM tab user WHERE userid=2;
 12     DBMS_OUTPUT.PUT_LINE('用户的编号为：' ||my.v_num) ;
 13     DBMS_OUTPUT.PUT_LINE('年龄为：' ||my.v_age) ;
 14  END;
 15  /
用户的编号为：2
年龄为：24
PL/SQL 过程已成功完成。
```

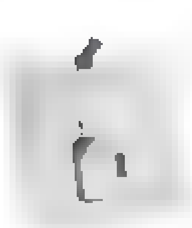
3．%ROWTYPE 类型

在 PL/SQL 中提供的%ROWTYPE 类型可以根据数据表中行的结构定义数据类型，并存储数据表中检索到的一行数据。例如：

```
SQL>  SET SERVEROUTPUT ON
SQL>  DECLARE
  2     row_user tab_user %ROWTYPE;
  3   BEGIN
  4     SELECT * INTO row_user
  5     FROM tab_user WHERE userid=2;
  6     DBMS OUTPUT.PUT LINE('用户的编号为：' ||row user.userid) ;
  7     DBMS_OUTPUT.PUT_LINE('用户的名称为：' ||row_user.username) ;
  8     DBMS_OUTPUT.PUT_LINE('性别为：' || row_user.usersex) ;
  9     DBMS_OUTPUT.PUT_LINE('年龄为：' || row_user.userage) ;
 10   END;
 11   /
用户的编号为：2
用户的名称为：dcyandly
性别为：女
年龄为：24
PL/SQL 过程已成功完成。
```

在上述代码中声明了一个%ROWTYPE 类型的变量，该变量的结构与表 tab_user 的结构完全相同，因此可以检索到的一行数据保存到该类型的变量中，并根据表中列的名称引用对应的数据。

10.6.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1197-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

10.7 PL/SQL 中游标使用的问题

10.7.1 问题描述

我声明了一个名称为 cursor_emp 的游标，该游标用于存储 scott.emp 表中的所有数据，目前使用 FOR 语句将存储在游标 cursor_emp 中 ename 列的数据循环读取，但执行之后出现以下错误，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     CURSOR cursor_emp IS SELECT * FROM scott.emp;      --声明游标
  3  BEGIN
  4     FOR current_cursor IN cursor_emp LOOP
  5         DBMS_OUTPUT.PUT_LINE('当前检索第'||cursor_emp%ROWCOUNT||'行：'||
  6         ename);
  7     END LOOP;
  8  END;
  9  /
ename);
*
第 6 行出现错误:
ORA-06550: 第 6 行, 第 1 列:
PLS-00201: 必须声明标识符 'ENAME'
ORA-06550: 第 5 行, 第 1 列:
PL/SQL: Statement ignored
```

由于初次接触游标，有很多迷惑的地方，请各位大哥大姐不吝指教，谢谢！

10.7.2 解决方法

PL/SQL 程序运行的错误指出第 6 行出现错误，这里使用的是 FOR 语句，因此需要使用 current_cursor.ename 的方式来获取游标中的数据，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
```

```
  2     CURSOR cursor_emp IS SELECT * FROM scott.emp;      --声明游标
  3  BEGIN
  4     FOR current_cursor IN cursor_emp LOOP
  5         DBMS_OUTPUT.PUT_LINE('当前检索第' || cursor_emp%ROWCOUNT || '行：' ||
  6         current_cursor.ename);
  7     END LOOP;
  8  END;
  9  /
当前检索第 1 行：SMITH
当前检索第 2 行：ALLEN
当前检索第 3 行：WARD
当前检索第 4 行：JONES
当前检索第 5 行：MARTIN
当前检索第 6 行：BLAKE
当前检索第 7 行：CLARK
当前检索第 8 行：SCOTT
当前检索第 9 行：KING
当前检索第 10 行：TURNER
当前检索第 11 行：ADAMS
当前检索第 12 行：JAMES
当前检索第 13 行：FORD
当前检索第 14 行：MILLER
PL/SQL 过程已成功完成。
```

10.7.3 知识扩展——游标

当使用 SELECT 语句检索结果时，返回的结果通常是多行记录，如果需要对多行记录中的一行数据单独进行操作，就需要使用游标。使用游标时主要有 4 个步骤，分别是：声明游标、打开游标、检索游标和关闭游标。对游标的基本操作主要有使用游标循环读取数据和使用游标对数据表中的数据进行更新和删除。下面将详细介绍这些操作。

1．声明游标

声明游标主要是定义一个游标名称来对应一条查询语句，声明游标的语法格式如下：

```
CURSOR cursor_name IS SELECT…
```

语法说明如下：

- **CURSOR**　表示游标关键字。
- **cursor_name**　表示需要定义的游标的名称。
- **SELECT…**　表示建立游标所用的查询语句。

例如，声明一个名称为 cursor_tabuser 的游标，用来查询 tab_user 表的所有信息，如下：

```
DECLARE
  CURSOR cursor_tabuser IS SELECT * FROM tab_user;  --声明游标
BEGIN
```

```
… /*执行语句部分*/
END;
/
```

2．打开游标

声明游标后必须将游标打开方可使用，打开游标需要使用 OPEN 语句，语法如下所示：

```
OPEN cursor_name;
```

上述语法中，cursor_name 表示游标的名称。

例如，打开声明的游标 cursor_tabuser，如下：

```
DECLARE
  CURSOR cursor_tabuser IS SELECT * FROM tab_user;  --声明游标
BEGIN
OPEN  cursor_tabuser;  --打开游标
… /*执行语句部分*/
END;
/
```

3．检索游标

将游标打开之后，使用 SELECT 语句查询的结果被临时存放到游标结果集中。如果需要获取从结果集中的数据，就需要检索游标。检索游标，实际上就是从结果集中获取单行数据并保存到定义的变量中，需要使用到 FETCH 语句，其语法如下：

```
FETCH cursor_name INTO variable1 , variable2,…;
```

在上述语法中，variable1，variable2 等是用来存放游标中相应字段数据的变量，其中变量的个数、顺序及类型要与游标中相应字段保持一致。

例如，定义一个%ROWTYPE 类型的变量 row_user，然后将使用 FETCH 语句提取游标中的数据放到变量 row_user 中。如下：

```
DECLARE
    CURSOR cursor_tabuser IS SELECT * FROM tab_user;
    row_user tab_user %ROWTYPE;
BEGIN
    OPEN cursor_tabuser;
    FETCH cursor_tabuser INTO row_user;
… /*其他执行语句部分*/
END;
```

4．关闭游标

游标使用完之后，必须使用 CLOSE 语句将其关闭，语法如下：

```
CLOSE cursor_name;
```

5. 简单游标循环

在游标中查询语句返回的是一个结果集，在结果集中可能包含多行数据。由于上面提到的检索数据检索的是单行数据，那么如何检索多行数据呢？实际上游标中的记录是循环读取的，没循环一次读取一行记录。

在介绍如何循环读取游标中的数据之前，首先来了解一下游标的属性，如下：

- ❑ **%FOUND** 返回布尔类型的值。用于判断最近一次读取记录时是否有数据行返回，如果有则返回 TRUE，否则返回 FALSE。
- ❑ **%NOTFOUND** 返回布尔类型的值，与%FOUND 相反。
- ❑ **%ISOPEN** 返回布尔类型的值。用于判断游标是否已经打开，如果已经打开则返回 TRUE，否则返回 FALSE。
- ❑ **%ROWCOUNT** 返回数字类型的值。用于返回已经从游标中读取的记录数。

例如，使用 LOOP 循环语句实现循环读取游标中的数据，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     CURSOR cursor_tabuser IS SELECT * FROM tab_user;  --声明游标
  3     row_user tab_user %ROWTYPE;                       --声明变量
  4  BEGIN
  5     OPEN  cursor_tabuser;                             --打开游标
  6     LOOP
  7         FETCH cursor_tabuser INTO row_user;           --检索游标
  8         EXIT WHEN cursor_tabuser%NOTFOUND;     --当游标无返回记录时退出循环
  9         DBMS_OUTPUT.PUT_LINE('当前检索第' || cursor_tabuser%ROWCOUNT || '
            行：' ||
 10         row_user.userid);          --显示当前检索行，以及记录中的 userid 值
 11     END LOOP;
 12     CLOSE cursor_tabuser;          --关闭游标
 13  END;
 14  /
当前检索第 1 行：1
当前检索第 2 行：2
PL/SQL 过程已成功完成。
```

6. 游标 FOR 循环

除了前面介绍了使用 LOOP 循环语句可以循环读取游标中的数据之外，还可以使用 FOR 循环语句。相对于 LOOP 语句，在 FOR 语句中设置的循环变量本身就存储了当前检索记录的所有列值，根本不需要手动打开和关闭游标，也不需要判断游标是否有返回记录，同样更不需要定义变量来接受记录值。

例如，使用 FOR 语句循环读取游标 cursor_tabuser 中的数据，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     CURSOR cursor_tabuser IS SELECT * FROM tab_user;  --声明游标
  3  BEGIN
```

```
  4     FOR current_cursor IN cursor_tabuser LOOP          --使用 FOR 循环
  5     DBMS_OUTPUT.PUT_LINE('当前检索第' || cursor_tabuser%ROWCOUNT || '行: ' ||
  6     current_cursor.userid);
  7  END LOOP;
  8  END;
  9  /
当前检索第 1 行: 1
当前检索第 2 行: 2
PL/SQL 过程已成功完成。
```

7. 使用游标更新或删除数据

使用游标可以更新和删除表中的数据。实现使用游标更新数据，则需要在声明游标时使用 FOR UPDATE 子句，然后便可以在 UPDATE 和 DELETE 语句使用 WHERE CURRENT OF 子句，从而实现对游标结果集中当前行对应的数据更新或者删除。

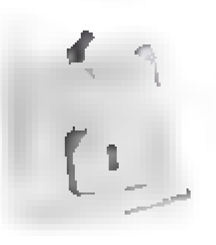
例如，创建带有 FOR UPDATE 子句的 CURSOR 语句，该游标用来检索 tab_user 表中的数据，如下：

```
SQL>  DECLARE
  2     CURSOR cur_user IS SELECT * FROM tab_user FOR UPDATE; --定义游标
  3     row_user tab_user %ROWTYPE;                           --定义变量
  4   BEGIN
  5     OPEN cur_user;                               --打开游标
  6     LOOP
  7         FETCH cur_user INTO row_user;            --提取游标数据赋给变量
  8         EXIT WHEN cur_user%NOTFOUND;
  9         IF row_user.userid=1 THEN  --当变量 row_user.userid 等于 1 时执行更新
 10             UPDATE tab_user SET username='anxin' WHERE CURRENT OF cur_user;
 11         END IF;
 12     END LOOP;
 13     CLOSE cur_user;                              --关闭游标
 14  END;
 15  /
PL/SQL 过程已成功完成。
```

在上述代码中定义了一个名称为 cur_user 的游标，该游标用于储存表 tab_user 中的数据；接着提取游标数据赋值给定义的%ROWTYPE 类型的变量。最后判断变量 row_user.userid 的值，等于 1 时执行更新的操作。

使用游标删除表中的数据和使用游标更新表中的数据类似，不同的是将 UPDATE 关键字换为 DELETE 关键字。这里不再举例介绍。

10.7.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1205-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1206-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

10.8 PL/SQL 中自定义异常的问题

10.8.1 问题描述

在 PL/SQL 中自定义异常可以这样：

```
MyException EXCEPTION;
```

为自定义异常 MyException 指定一个错误号，如下：

```
PRAGMA EXCEPTION_INIT(MyException,-20001);
```

那么请问，如何才能使 MyException 返回我自己指定的错误信息呢？请各位帮帮忙，谢谢！

10.8.2 解决方法

如果要显示自己指定的错误信息，需要使用 RAISE 关键字，具体的实现如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  MyException EXCEPTION;
  3  PRAGMA EXCEPTION INIT(MyException,-20001);
  4  BEGIN
  5  RAISE MyException;
  6  EXCEPTION WHEN MyException THEN
  7  DBMS OUTPUT.PUT LINE('我的错误信息');
  8  END;
  9  /
我的错误信息
PL/SQL 过程已成功完成。
```

10.8.3 知识扩展——异常处理

异常就是 PL/SQL 程序在执行时出现的错误。当产生异常时，程序员对可能出现的异常进行控制的方式被称为异常处理。产生异常的原因有很多，例如程序本身出现的逻辑错误，在程序中一旦出现异常，必须进行处理，否则程序就会终止。

异常处理的一般语法如下：

```
EXCEPTION
  WHEN exception1 TEHN
    Statements;
…
```

```
WHEN exceptionn TEHN
  Statements;
WHEN OTHERS THEN
  Statements;
```

当 PL/SQL 程序检测到异常 exception1 时，程序就会转入相应异常处理代码段 Statements 中进行处理。其中，WHEN OTHERS THEN 子句指异常如果不在前面所列的异常处理之中，将进入 OTHERS 异常处理程序段。

Oracle 系统中的异常分为两种，分别是：系统预定义异常和用户自定义异常。

1. 预定义异常

预定义异常，是指 Oracle 系统为一些常见错误定义好的异常。这些异常无需声明，当程序出现错误时，Oracle 系统会自动触发，这里只需添加相应的异常处理即可。常见的 Oracle 系统预定义异常如表 11-5 所示。

表 11-5　常见的 Oracle 系统预定义异常

错误信息	异常错误名称	说　明
ORA-0001	Dup_val_on_index	试图破坏一个唯一性限制
ORA-0051	Timeout-on-resource	在等待资源时发生超时
ORA-0061	Transaction-backed-out	由于发生死锁事务被撤销
ORA-1001	Invalid-CURSOR	试图使用一个无效的游标
ORA-1012	Not-logged-on	没有连接到 Oracle
ORA-1017	Login-denied	无效的用户名/口令
ORA-1403	NO_DATA_FOUND	SELECT INTO 没有找到数据
ORA-1422	TOO_MANY_ROWS	SELECT INTO 返回多行
ORA-1476	Zero-divide	试图被零除
ORA-1722	Invalid-NUMBER	转换一个数字失败
ORA-6500	Storage-error	内存不够引发的内部错误
ORA-6501	Program-error	内部错误
ORA-6502	Value-error	转换或截断错误
ORA-6504	Rowtype-mismatch	主变量和游标的类型不兼容
ORA-6511	CURSOR-ALERADY-OPEN	试图打开一个已经打开的游标时，将产生这种异常
ORA-6530	Access-INTO-null	试图为 null 对象的属性赋值

例如，在 PL/SQL 程序中使字符串转换为数字将会出现如下异常：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  num VARCHAR2(10) := 'dcyandly' ;
  3  BEGIN
  4  DBMS_OUTPUT.PUT_LINE('测试异常发生前执行') ;
  5  DBMS OUTPUT.PUT LINE(CAST(num AS NUMBER)) ;
  6  DBMS OUTPUT.PUT LINE('测试异常发生后执行') ;
  7  END ;
```

```
  8  /
测试异常发生前执行
DECLARE
*
第 1 行出现错误:
ORA-06502: PL/SQL: 数字或值错误 :  字符到数值的转换错误
ORA-06512: 在 line 5
```

从执行结果可以看到，Oracle 抛出 ORA-06502 异常，提示字符到数值的转换错误。从异常代码可以得知异常的名称，因此错误代码 ORA-06502 对应的名称是 Value-error。

为了程序不被终止，在 PL/SQL 程序块中添加异常处理的代码，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     num VARCHAR2(10) := 'dcyandly' ;
  3  BEGIN
  4     DBMS_OUTPUT.PUT_LINE('测试异常发生前执行') ;
  5     DBMS OUTPUT.PUT LINE(CAST(num AS NUMBER)) ;
  6     EXCEPTION
  7     WHEN VALUE ERROR THEN
  8     DBMS_OUTPUT.PUT_LINE('异常提示：该字符串不能转换为有效数字！') ;
  9     DBMS_OUTPUT.PUT_LINE('测试异常发生后执行') ;
 10  END ;
 11  /
测试异常发生前执行
异常提示：该字符串不能转换为有效数字！
测试异常发生后执行
PL/SQL 过程已成功完成。
```

从 PL/SQL 程序块执行的结果可以看出，当异常被处理后，Oracle 不再提示异常信息，并且一直向下执行处理异常之后的内容。

2．用户自定义异常

除了上面介绍的系统预定义异常之外，用户还可以根据需求，为实现具体的业务逻辑自定义相关的异常。

用户自定义异常需在 PL/SQL 程序块的声明部分中进行声明，其语法如下：

```
exception name  EXCEPTION;
PRAGMA  EXCEPTION_INTO(exception_name,exception_no);
```

在上述语法中，exception name 表示用户自定义异常的名称，PRAGMA 关键字表示可以让异常名和异常号联系起来。

用户自定义异常需要使用 RAISE 语句显示触发的异常。例如，在 QQ 账号中以自己不能删除自己为例来说明自定义异常的使用，如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
```

```
  2  e_qqid EXCEPTION;                          --定义异常
  3  PRAGMA EXCEPTION_INIT(e_qqid,-1235);
  4  qqid NUMBER:=1;
  5  BEGIN
  6  IF (qqid=1) THEN
  7  RAISE e_qqid;                              --显示触发异常
  8  END IF;
  9  EXCEPTION
 10  WHEN e_qqid THEN
 11  DBMS_OUTPUT.PUT_LINE('自己不能删除自己');
 12  END;
 13  /
自己不能删除自己
PL/SQL 过程已成功完成。
```

在上述代码中，定义了一个名称为 e_qqid 的异常，并为异常指定异常号-1235。接着声明一个名称为 qqid 的变量并为其赋值 1，然后判断，当变量 qqid 等于 1 时引发 e_qqid 异常。

10.8.4 网络课堂

视频教学：http://school.itzcn.com/video-vid-1207-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1208-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1209-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-475-1-1.html

第 11 章　PL/SQL 高级应用

PL/SQL 是 SQL 的一种扩展语言，本章将介绍 PL/SQL 的一些高级应用，包含存储过程、触发器和程序包等内容。其中存储过程是一种命名的 PL/SQL 程序块；触发器类似于事件执行机制；程序包就是组合在一起的相关对象的集合；函数是用于处理一些数据。

本章将为大家详细介绍在 PL/SQL 中使用、创建和操作存储过程、函数、程序包和触发器等知识。

11.1　PL/SQL 存储过程出错

11.1.1　问题描述

存储过程是 PL/SQL 程序块，并且使用存储过程可以设置输入输出参数。存储过程没有返回值，在执行存储过程时需要使用 EXECUT 命令或者其他内部命令来调用，而不可以直接执行。可是在创建存储过程时出现错误，代码如下所示。

各位学哥学姐们来帮我看下这个问题，初学 PL/SQL 存储过程，随便编写了一段过程，要求将方法内的字符串输出，可是在执行时出现编译错误，请各位帮忙解决下？代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE show user
  2  BEGIN
  3  DBMS_OUTPUT.PUT_LINE("欢迎使用存储过程");
  4  END;
  5  /
警告：创建的过程带有编译错误。
```

上述代码创建了名为 show_user 的存储过程，在该存储过程中使用 DBMS_OUTPUT.PUT_LINE 语句输出一段话。

11.1.2　解决方法

存储过程和普通的 PL/SQL 创建方式不同，在创建存储过程时 BEGIN 关键字前必须使用 IS 或 AS 关键字，正确的创建代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE show user
  2  IS
  3  BEGIN
  4  DBMS OUTPUT.PUT LINE('欢迎使用存储过程');
```

```
  5  END;
  6  /
过程已创建。
```

11.1.3　知识扩展——创建过程与修改

存储过程是由流控制和 SQL 语句书写的过程，这个过程经编译和优化后存储在数据库服务器中，使用时只要调用即可。存储过程需要使用 CREATE PROCEDURE 语句来创建。如果在创建时使用 OR REPLACE 语句，那么存在相同名称的存储过程时将会替换已有存储过程，创建语法如下所示。

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] datatype [,…])]
{IS | AS}
BEGIN
procedure_body
END;
/
```

上述语法中，procedure_name 表示创建的存储过程名称；OR REPLACE 表示如果存在此存储过程则将原有存储过程替换；parameter 表示参数，在存储过程中可以一个或者多个参数，中间使用逗号隔开；procedure_body 就表示存储过程的功能操作；在创建完存储过程之后，在 END 后面使用“/”执行创建操作。

存储过程创建完成后，存储过程中的程序并没有被执行。此时，还需要使用其他语句进行调用，详细语法如下。

```
EXEC[UTE] procedure_name [ ( parameter [ , … ] ) ] ;
```

在调用存储过程的语法中，procedure_name 表示要被执行的存储过程名称；parameter 表示参数，如果在存储过程中设置了参数，那么可以在这里将参数传入存储过程内。

11.1.4　触类旁通



创建存储过程出现 ORA-00955 错误？

网络课堂：http://bbs.itzcn.com/thread-16829-1-1.html

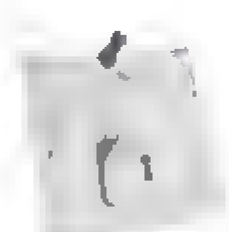
使用 CREATE PROCEDURE 语句创建存储过程时，提示名称已经存在的错误。如何将已经存在的存储过程替换为新创建的存储过程。以下代码则是创建 show_user 过程出现错误的代码。

```
SQL> CREATE PROCEDURE show_user
  2  IS
  3  BEGIN
  4  DBMS_OUTPUT.PUT_LINE('再次创建存储过程');
```

```
  5  END;
  6  /
CREATE PROCEDURE show user
                *
第 1 行出现错误:
ORA-00955: 名称已由现有对象使用
```

在创建存储过程时使用 OR REPLACE 语句，这样如果当前创建的存储过程存在，那么 Oracle 将会把该存储过程替换掉。因此该语句也可以作为修改存储过程来使用，用户只需要在创建存储过程时 OR REPLACE 语句，并且把创建的存储过程名称和要被修改的过程名称相同即可。

11.1.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1210-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16828-1-1.html

11.2 是否可以删除不必要的存储过程

11.2.1 问题描述

有时你可能会发现自己的数据库中存在着很多没用的存储过程，这样给数据库的管理带来很多麻烦，是否可以通过使用命令将不必要的存储过程删除掉。

11.2.2 解决方法

在 PL/SQL 中，要删除存储过程非常简单，只需要一条命令即可。它的删除方法和删除表的方法相似，不过在这里使用的是 PROCEDURE 关键字，详细删除代码如下所示。

```
SQL> connect system/tiger
已连接。
SQL> DROP PROCEDURE show user;
过程已删除。
```

在执行删除时，操作用户也必须具备这样的权限，删除的存储过程必须存在。

11.2.3 知识扩展——删除过程

在操作存储过程时，可以使用 DROP PROCEDURE 命令来删除不需要的存储过程。其删

除语法如下所示。

```
DROP PROCEDURE procedure name
```

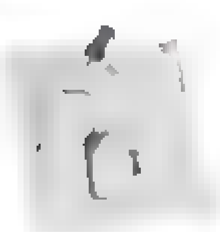
例如在下面实例代码中，首先通过使用 CREATE 语句创建一个名为 out name 的存储过程，在该过程中输出用户的姓名，代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE out name
  2  IS
  3  BEGIN
  4  DBMS OUTPUT.PUT LINE("张三");
  5  END;
  6  /
过程已创建。
```

过程创建完成后，由于是用于测试，所以在测试完成后需要对该过程进行删除，那么删除的具体代码如下。

```
SQL> DROP PROCEDURE out name;
过程已删除。
```

11.2.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1214-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16830-1-1.html

11.3 如何使一个存储过程显示不同结果

11.3.1 问题描述

使用存储过程的最大好处是可以减少代码量，这是我们众所周知的。在通过用户名称查询用户信息的存储过程中，如果要想查询多个用户的信息是不是需要创建多个存储过程？我想不应该，Oracle 肯定会为我们提供更好的解决方案。请大家帮我解惑。

11.3.2 解决方法

在使用存储过程时，可以为存储过程设置参数。当用户执行存储过程时，传入不同的参数所执行的操作也不相同。例如以下代码则是通过传入不同的用户名称，显示不同的用户信息。代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE show user(uname VARCHAR2,uage NUMBER)
```

```
  2  IS
  3  BEGIN
4  DBMS OUTPUT.PUT LINE(uname||uage);
  5  END;
  6  /
过程已创建。
SQL> SET SERVEROUTPUT ON;
SQL> EXEC show_user('张三',20);
张三20
PL/SQL 过程已成功完成。
```

上述代码创建了show_user存储过程，并且为该存储过程设置了uname和uage两个参数。在过程体中将用户输入的参数输出。

11.3.3 知识扩展——过程的参数传递

在创建存储过程时，为了方便程序的应用、减少代码量等，通常会为存储过程添加参数。参数是一种向程序体输入和输出数据的机制，因此参数也可以在存储过程中使用，不过参数传递有按位置传递、按名称传递和混合方式传递三种。

1. 位置传递

按位置传递参数是指调用时的实参数据类型、个数与形参在相应位置上要保持一致。如果位置没有对应，那么将产生错误。例如以下代码是运行show_user过程参数传递。

```
SQL> SET SERVEROUTPUT ON;
SQL> EXEC show_user('张三',20);
张三20
PL/SQL 过程已成功完成。
```

这样传递参数的方式会经常使用到，使用时确保对应位置的参数类型和数量匹配即可。

2. 名称传递

参数以名称传递的方式来设置，可以避免使用位置传递所引发的问题。按名称传递不需要估计位置是否正确，只需要确保对应的参数的数据类型相匹配即可，例如以下代码同样使用show_user存储过程。

```
SQL> SET SERVEROUTPUT ON;
SQL> EXEC show_user(uage=>22,uname=>'李四');
李四22
PL/SQL 过程已成功完成。
```

使用名称传递时，参数名称和参数之间要使用“=>”符号进行关联。

3. 混合传递

所谓的混合传递就是将位置传递和名称传递两种传递方式结合使用。这样的参数传递方式适合过程具有可选参数的情况。以下则是使用方法。

```
SQL> SET SERVEROUTPUT ON;
SQL> EXEC show user('王二',uage >25);
李四 22
PL/SQL 过程已成功完成。
```

11.3.4 触类旁通



运行带参存储过程出错？

网络课堂：http://bbs.itzcn.com/thread-16832-1-1.html

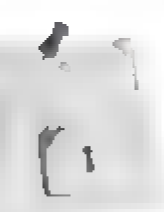
创建存储过程时，为存储过程设置了相应的参数可以提高程序的运行效率。可是在运行存储过程时，却出现了如下错误。

```
SQL> EXEC show_user(uage=>25,'小强');
BEGIN show_user(uage=>25,'小强'); END;
                            *
第 1 行出现错误:
ORA-06550: 第 1 行, 第 26 列:
PLS-00312: 一个定位相关参数没有说明其相关性
ORA-06550: 第 1 行, 第 7 列:
PL/SQL: Statement ignored
```

在运行带参的存储过程时使用混合参数传递方式时和普通的位置传递参数非常相似，它除了必须遵守位置传递的规则外，还需要遵守名称传递方式规则。在你编写的代码中，只遵守了名称传递方式，而忽略了位置传递规则，其正确的代码如下。

```
SQL> EXEC show_user('小强',uage=>25);
小强 25
PL/SQL 过程已成功完成。
```

11.3.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1211-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16831-1-1.html

11.1 创建过程语法错误

11.4.1 问题描述

在创建存储过程时，为了程序的需要，声明了 uname、uid 和 ustar 三个参数，其中 uname 参数用于作为查询数据的条件参数；uid 表示查询数据用户的 ID 信息；ustar 则表示输出用户

的 USERSTAR 信息。可是在创建过程时出现语法错误。代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE show user
  2  (uname VARCHAR2,uid NUMBER,ustar NUMBER)
  3  IS
  4  BEGIN
  5  SELECT userid,userstar INTO uid,ustar FROM userinfo
  6  WHERE username=uname;
  7  END;
  8  /

警告：创建的过程带有编译错误。
```

上述案例中使用 SELECT INTO 语句将查询 userinfo 表中的 userid 和 userstar 信息并存储在 uid 和 ustar 参数中输出。可是在执行过程中出现编译错误。请大家指点迷津。

11.4.2　解决方法

如果创建的存储过程中需要输出或者输入参数值，那么必须使用特殊的参数模式，而在上述错误代码中，可以看出需要将 uname 参数传入值，而 uid 和 ustar 则需要输出值。因此在定义参数时需要使用 IN 和 OUT 关键字，正确的代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE show user
  2  (uname VARCHAR2,uid OUT NUMBER,ustar OUT NUMBER)
  3  IS
  4  BEGIN
  5  SELECT userid,userstar INTO uid,ustar FROM userinfo
  6  WHERE username=uname;
  7  END;
  8  /

过程已创建。
```

在代码中，为输入参数 uname 定义 IN 关键字，由于参数的默认模式为 IN，因此在这里可以将 IN 关键字省略。然后为 uid 和 ustar 定义 OUT 关键字用来输出查询信息。

11.4.3　知识扩展——过程的参数模式

存储过程的参数模式的功能是描述当前参数的行为和作用。在 PL/SQL 中参数模式分为输入模式（IN）、输出模式（OUT）和输入输出模式（IN OUT）。在设置参数模式时，只需要将关键字添加在参数和参数类型之间，即可改变该参数的模式。以下则是这 3 种模式的详细介绍。

1．输入参数模式（IN）

参数输入模式是由关键字 IN 来决定的，设置该模式后，参数可以通过该模式输入到过程体内，提供过程的操作使用。默认情况下在过程中定义的所有参数默认模式都为输入参数，因此把参数模式设置为输入时通常是不填写关键字的。例如以下代码是正确的定义输入参数

模式。

```
CREATE PROCEDURE procedure name(parameter name IN datatype)
IS
BEGIN
procedure body
END
```

2．输出参数模式（OUT）

定义输出参数模式时需要使用 OUT 关键字，该模式的参数的作用是将过程体内的程序运行结果输出。而在设置该模式参数时，OUT 关键字是必须存在的。该模式的参数正确的编写代码如下所示。

```
CREATE PROCEDURE procedure name(parameter name OUT datatype)
IS
BEGIN
procedure body
END
```

3．输入输出参数模式（IN OUT）

输入输出模式的参数的功能就相对完善了，该模式参数需要使用 IN OUT 关键词来定义，它可以使参数既可以向过程体内输入信息，也可以将过程体内数据输出，该模式参数创建方式如下所示。

```
CREATE PROCEDURE procedure name(parameter name IN OUT datatype)
IS
BEGIN
procedure body
END
```

11.4.4 知识扩展——参数默认值

参数默认值就相当于参数初始值，它主要作用是：在使用该参数时，如果没有为该参数指定值，那么该参数的值就为参数默认值。参数默认值的声明语法如下所示。

```
parameter_name parameter_type {[DEFAULT | :=]}value
```

在上述语法中，parameter_name 表示参数名称；parameter_type 表示参数类型；DEFAULT 则表示设置默认值用到的关键字；value 表示设置的默认值。设置默认值时需要使用:=符号将值赋给参数。

11.4.5 触类旁通



运行带参存储过程出错？

网络课堂：http://bbs.itzcn.com/thread-16834-1-1.html

新创建的带参数存储过程用于输出用户输入的用户名称，在该过程中有 uname 参数，该参数类型为 VARCAHR2，当使用 EXEC 语句运行 show_user 过程时出现下面的异常信息，如何解决？

```
SQL> CREATE OR REPLACE PROCEDURE show_user
  2  (uname VARCHAR2)
  3  IS
  4  BEGIN
  5  DBMS_OUTPUT.PUT_LINE(uname);
  6  END;
  7  /
过程已创建。

SQL> SET SERVEROUTPUT ON;
SQL> EXEC show_user();
BEGIN show_user(); END;
      *
第 1 行出现错误:
ORA-06550: 第 1 行, 第 7 列:
PLS-00306: 调用 'SHOW_USER' 时参数个数或类型错误
ORA-06550: 第 1 行, 第 7 列:
PL/SQL: Statement ignored
```

在上述存储过程中，你创建了带参数的存储过程，并且设置了 uname 参数。可是在运行该过程时为给该过程设置参数值，因此就出现了这样的参数错误提示。

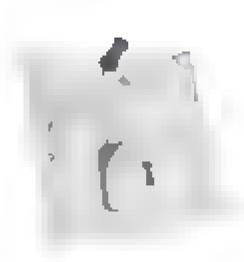
解决办法可以为该过程中的参数设置默认值，如果在运行时为给该参数设置参数值，那么系统将会自动调用参数默认值，详细代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE show_user
  2  (uname VARCHAR2 :='为设置名称')
  3  IS
  4  BEGIN
  5  DBMS_OUTPUT.PUT_LINE(uname);
  6  END;
  7  /
过程已创建。

SQL> EXEC show_user();
为设置名称
PL/SQL 过程已成功完成。
```

在代码中则是为 uname 参数设置默认值为“为设置默认值”，在运行该过程时系统就将默认参数传入过程中，因此输出的结果就为“为设置默认值”。

11.4.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-1212-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1213-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16833-1-1.html

11.5 如何在 PL/SQL 脚本中调用自定义函数

11.5.1 问题描述

在其他编程语言中，函数用于返回特定的程序数据，这样能够方便程序的制作，也减少了代码的编写量。在前面学习了使用存储过程可以优化程序和减少代码量，除此之外在 PL/SQL 中是否还存在其他模块具有这些功能呢？

11.5.2 解决方法

在 PL/SQL 的高级应用中也存在“函数”的概念，它的作用和功能和程序语言中的函数几乎相似，而且和存储过程的功能也非常相似。不过函数中包含 RETURN 子句，用来进行数据操作，并且函数的调用只能在一个表达式中。以下代码则是创建了个简单的函数的创建代码。

```
SQL> CREATE FUNCTION select user
  2  RETURN VARCHAR2
  3  AS
  4  uname USERINFO.USERNAME%TYPE;
  5  BEGIN
  6  SELECT username INTO uname FROM userinfo;
  7  RETURN uname;
  8  END;
  9  /
函数已创建。

SQL> DECLARE
  2  uname VARCHAR2(50);
  3  BEGIN
  4  uname :=select user;
  5  DBMS OUTPUT.PUT LINE(uname);
  6  END;
  7  /
张三
PL/SQL 过程已成功完成。
```

在上述代码中创建了名为 select user 的函数，并且该函数体用于执行 SELECT 查询语句，将 userinfo 表中的用户名称读取出来绑定在 uname 变量中，然后返回。最后调用该函数输出结果。

11.5.3 知识扩展——函数的基本操作

如果需要使用函数，那么第一步必须先创建函数。函数的创建方法和存储过程相似，不过需要使用 CREATE FUNCTION 语句来完成。在创建函数时也可以为函数设置输入模式参数、输出模式参数和输入输出默认参数。这些参数的模式作用和过程相同，以下代码则是创建函数的语法结构。

```
CREATE [OR REPLACE] FUNCTION function name
[(parameter {IN | OUT | IN OUT} datatype)]
RETURN datatype
{ IS | AS }
BEGIN
    function body
END
```

上述语法中 function_name 表示创建的函数名称；parameter 表示为函数设置的参数名称；datatype 表示该函数的返回类型，必须有该属性的配置。function_body 表示函数体，在函数中所实现的功能和 PL/SQL 语句块都存放在这里。

函数除了创建之外，还可以进行删除操作，以下语法则是删除函数语法结构。

```
DROP FUNCTION function_name;
```

删除时需要使用 DROP FUNCTION 语句来完成，其中 function_name 表示被删除的函数名称。

11.5.4 触类旁通



无法成功创建带参函数？

网络课堂：http://bbs.itzcn.com/thread-16836-1-1.html

为了创需的需要，创建带参数的函数，在函数体内将用户输入的内容打印出来，可是在创建时出现编译异常，我已经检查了很多遍，代码没有错误，希望能够解决下这个问题。详细代码如下所示。

```
SQL> CREATE FUNCTION show uname(uname VARCHAR2)
  2  RETURN VARCHAR2
  3  IS
  4  BEGIN
  5  DBMS OUTPUT.PUT LINE(uname);
  6  END;
  7  /

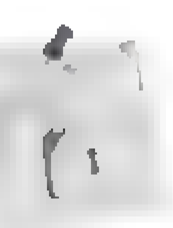
警告：创建的函数带有编译错误。
```

如果创建带参数的函数，你的代码编写是没有错误的。可是由于你程序的参数性质是输出模式，而你在声明参数时没有指定它的参数模式设置为输出，因此在创建函数时就出现错误提示信息，正确的编写方式如下所示。

```
SQL>  CREATE FUNCTION show uname(uname OUT VARCHAR2)
  2  RETURN VARCHAR2
  3  IS
  4  BEGIN
  5  DBMS OUTPUT.PUT LINE(uname);
  6  END;
  7  /

函数已创建。
```

11.5.5　网络课堂



视频教学：http://school.itzcn.com/video-vid-1215-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16835-1-1.html

11.6　如何创建 PL/SQL 程序包

11.6.1　问题描述

在程序中包有着非常重要的作用，可以有效地隐藏信息，实现集成化的模块程序设计，有利于程序的维护，而在 PL/SQL 中是否存在包的概念？如果存在，如何去创建 PL/SQL 程序包？它的作用是否和其他程序中包的作用相同？

11.6.2　解决方法

在 PL/SQL 中也存在包的概念，而且在 PL/SQL 中包也是由包头（包规范）和包体两部分组成。因此在创建包时需要先创建包头，然后创建包体。详细创建代码如下所示：

```
SQL> CREATE PACKAGE package user
  2  AS
  3  PROCEDURE show user(uname VARCHAR2,uage NUMBER);
  4  END;
  5  /

程序包已创建。
```

上述代码是成功创建了名为 package user 的包头，并且在该包头里声明了存储过程

show_user。接下来创建包体，并且在包体内实现 show_user 的存储，详细代码如下所示。

```
SQL> CREATE PACKAGE BODY package_user
  2  AS
  3  PROCEDURE show_user
  4  (uname VARCHAR2,uage NUMBER)
  5  AS
  6  BEGIN
  7  DBMS_OUTPUT.PUT_LINE(uname||uage);
  8  END show_user;
  9  END package_user;
 10  /

程序包体已创建。
```

在该包体内实现了 show_user 过程功能，将用户输入内容进行打印，这样就将该过程创建在了 package_user 包内，以下代码则是执行调用该表信息。

```
SQL> SET SERVEROUTPUT ON;
SQL> EXECUTE package_user.show_user('段韶治',25);
段韶治 25

PL/SQL 过程已成功完成。
```

11.6.3 知识扩展——创建包

包是由包头和包体组合而成的，在创建包时包头是必不可少的，包体则是可有可无的，但是实现代码都是存放在包体内的。本节将会为大家介绍如何创建一个包，并且在包中添加过程和函数，提供其他编程人员可以重复使用这些过程或者函数代码。

1．创建包头

包头也就是规范，包的规范列出了可用的过程、函数、类型和对象，而包规范通常是不可以编写包的实现代码，在创建包头时需要使用 CREATE PACKAGE 语句，详细语法如下所示。

```
CREATE [OR REPLACE] PACKAGE package_name
{ IS | AS}
   package_specification
END package_name;
```

上述语法中，package_name 表示创建的包名称；package_specification 表示在包中需要公用的过程、函数类型或者对象等结构。

2．创建包体

包体则是程序的实现体，将程序实现代码编写在这里。凡是在包头没有出现的项目，在包体内则表示私有对象只能在包体内使用和调用。创建包体需要使用 CREATE PACKAGE

BODY 语句，详细创建语法如下所示。

```
CREATE [ OR REPLACE ] PACKAGE BODY package name
{ IS ] AS }
    package body;
END package name;
```

上述语法中，package name 表示指定的包名称，必须和创建的包头名称相匹配。package body 表示包的实现体，将过程或者函数的实现代码编写 package_bod 的位置。

11.6.4 知识扩展——包的其他操作

在 PL/SQL 中，包的操作和存储过程函数等操作功能非常相似，例如包也可以进行修改和删除等操作。下面介绍下对包的删除和修改的语法。

1．修改包

包的修改方法和过程的修改方法相同，只需要在创建包时添加 OR REPLACE 选项就可以把之前创建好的包覆盖为当前的信息。以下语法则是修改包头的语法代码，同样修改包体也是相同的方法。

```
CREATE OR REPLACE PACKAGE package name
{ IS | AS}
    package specification
END package_name;
```

2．删除包

删除包需要使用 DROP PACKAGE 语句，其删除语法如下所示。

```
DROP PACKAGE package_name;
```

其中语法中 package_name 表示被删除的包的名称。

11.6.5 基础知识——系统预定义包

在 Oracle 数据库中，系统已经定义好了一些包，这些包都是以 DBMS 或 UTL_开头，这些包可以在很多编程语言中进行调用。在下表中则表 11-1 列出了系统定义好的包。

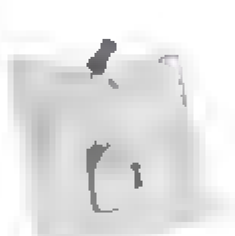
表 11-1 Oracle 系统默认包

[illegible]	说明
DBMS_ALERT	用于当数据改变时，使用触发器向应用发出警告
DBMS_DDL	用于访问 PL/SQL 中不允许直接访问的 DDL 语句
DBMS_Describe	用于描述存储过程与函数 API
DBMS_Job	用于作业管理
DBMS_Lob	用于管理 BLOB，CLOB，NCLOB 与 BFILE 对象
DBMS_OUTPUT	用于 PL/SQL 程序终端输出

续表

[illegible]	[illegible]
DBMS_PIPE	用于数据库会话使用管道通信
DBMS_SQL	用于在 PL/SQL 程序内部执行动态 SQL
UTL_FILE	用于 PL/SQL 程序处理服务器上的文本文件
UTL_HTTP	用于在 PL/SQL 程序中检索 HTML 页
UTL_SMTP	用于支持电子邮件特性
UTL_TCP	用于支持 TCP/IP 通信特性

11.6.6　网络课堂



视频教学：http://school.itzcn.com/video-vid-1216-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16837-1-1.html

11.7　创建的表如何才能自动填充 ID 列

11.7.1　问题描述

在前面学习过使用 SQL 语句创建表，创建表并不难，可是创建 Oracle 表时主键无法直接生成，需要其他功能语句来辅导主键自动生成值。而它的实现思路则是当用户向表内插入数据时，触动某个语句将创建好的序列中的值插入到当前新插入的数据 ID 列中。那么如何创建触发机制。

11.7.2　解决方法

在 Oracle 中有触发器的概念，通过触发器可以实现这个功能。首先创建一个序列生成动态的数值号，详细代码如下所示。

```
SQL> CREATE SEQUENCE "se_users"
  2  MINVALUE 1 MAXVALUE 999999
  3  INCREMENT BY 1
  4  START WITH 1
  5  CACHE 20
  6  NOORDER CYCLE;

序列已创建。
```

在触发器中，系统将会自动生成从 1 到 999999 的值。

下面则是创建一个名为 TR_USERS 的触发器，在用户向表中插入数据之前触发，从而将

生成好的序列的值写入主键中，这样在插入记录时就会自动生成主键了。详细代码如下所示。

```
SQL> CREATE OR REPLACE TRIGGER tr users
  2  BEFORE INSERT ON users
  3  FOR EACH ROW
  4  WHEN (NEW.ID IS NULL)
  5  BEGIN
  6  SELECT se users.NEXTVAL INTO :NEW.ID FROM DUAL;
  7  END;
  8  /
触发器已创建
```

上述代码在执行插入操作时会触发触发体内的程序，将产生的数据插入到新生成的 ID 列中。

11.7.3 知识扩展——触发器的类型

触发器就是当数据库表或数据库发生变化时自动执行的代码机制。它和过程函数有所相似之处，都具有声明和处理部分。唯一不相同的就是触发器不可以直接运行，必须通过某种操作来触发运行。

触发器也有不同的触发类型例如：DML 触发器、替代触发器、系统触发器和 DDL 触发器类型。下面则是触发器类型的详细介绍。

- **DML 触发器**

DML 触发器就是在执行 INSERT、DELETE 和 UPDATE 语句时自动调用执行触发器代码块。其中 DML 触发器又分为 DML 事件执行前触发（BEFORE）和 DML 事件执行后触发（AFTER）两种触发器。除此之外，DML 触发器还分行触发和语句触发。行触发器针对语句所影响的每一行都触发一次，而语句级触发器则针对某一条语句触发一次。

- **替代触发器**

这种触发器就能够替代原始的触发动作，该触发器扩展了视图更新的类型。对于每一种触发动作（INSERT、UPDATE 或 DELETE），每一个表或视图只能有一个 INSTEAD OF 触发器。

- **系统触发器**

系统触发器是指基于 Oracle 系统事件（如 logon 和 startup）所建立的触发器。通过这种触发器可以跟踪系统或数据库的变化。

- **DDL 触发器**

DDL 触发器是基于 CREATE、ALTER 或 DROP 等语句的触发器，它可以在这些语句执行前后来触发。

11.7.4 知识扩展——触发器的基本操作

触发器的操作与函数、过程、包等的操作大同小异。要想使用触发器必须创建触发器，除此之外触发器也可以进行删除、修改和查看等功能。下面介绍下在 PL/SQL 中如何对触发

器进行操作。

1．创建触发器

创建触发器需要使用CREATE TRIGGER语句，在创建时需要设置触发的时间和触发器的执行动作。创建语法如下所示。

```
CREATE TRIGGER trigger name
{BEFORE | AFTER |INSTEAD OF} trigger event
ON {table name | view name}
[FOR EACH ROW]
BEGIN
    trigger body
END trigger_name ;
```

语法中，trigger_name 表示创建的触发器名称；BEFORE | AFTER |INSTEAD OF 中BEFORE表示触发器在触发事件执行之前被激活，AFTER表示触发器在触发事件执行之后被激活，INSTEAD OF表示用触发器中的事件代替触发事件执行；trigger_event表示激活触发器的事件（INSERT、UPDATE和DELETE等）；ON table_name | view_name 指定DML触发器所针对的表。如果是INSTEAD OF触发器，则需要指定视图名（view_name）；如果是DDL触发器或系统事件触发器，则使用ON DATABASE；FOR EACH ROW表示触发器是行级触发器，如果没有此选项，则默认是语句级触发器。

2．修改触发器

修改触发器需要使用OR REPLACE选项，在过程函数中都有这样的选项。使用该选项时，创建的名称和被修改触发器的名称要相同，那么系统将会把以前的触发器覆盖为新创建的触发器。

```
CREATE [OR REPLACE] TRIGGER trigger_name
```

3．删除触发器

删除触发器需要使用DROP TRIGGER语句，语法如下所示。

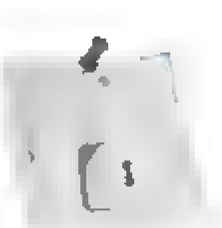
```
DROP TRIGGER trigger_name;
```

语法中trigger_name表示被删除的触发器名称。

4．查看触发器

在Oracle数据库中，每创建一个触发器都将会在与触发器有关的字典内保存数据。其中与触发器有关的USER_TRIGGER、ALL_TRIGGER和DBA_TRIGGER三个数据字典。其中USER_TRIGGERS存放当前用户的所有触发器，ALL_TRIGGERS存放当前用户可以访问的所有触发器，DBA_TRIGGERS存放数据库中的所有触发器。用户可以查询这些数据字典来获取当前的触发器信息。

11.7.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1217-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16838-1-1.html

11.8 如何获取修改前和修改后的值

11.8.1 问题描述

在对数据库表数据进行修改时，很多用户都习惯将修改前和修改后的值进行对比确保是否已经成功的修改了数据。那么在 Oracle 数据中是否有好的方法来获取修改前和修改后的值呢？

11.8.2 解决方法

在 Oracle 中可以使用触发器来实现这个功能，在触发器中有 OLD 和 NEW 关键两个关键字，这两个关键字则可以获取数据修改前和修改后的值。例如以下代码。

```
SQL> CREATE OR REPLACE TRIGGER trigger users
  2  BEFORE UPDATE ON users
  3  FOR EACH ROW
  4  DECLARE
  5  oldage NUMBER;
  6  newage NUMBER;
  7  BEGIN
  8  newage :=:NEW.userage;
  9  oldage :=:OLD.userage;
 10  DBMS_OUTPUT.PUT_LINE('修改前'||oldage||'修改后'||newage);
 11  END;
 12  /

触发器已创建
```

上述代码中创建了名为 trigger_user 的触发器，该触发器在对 users 表进行 UPDATE 操作时激活。然后声明修改前和修改后两个 NUMBER 类型变量。在触发体内通过使用 OLD 和 NEW 关键字获取 users 表中的 userage 列修改前和修改后的值，并赋予给 oldage 和 newage 两个变量，最后进行输出。

接下来执行 UPDATE 操作修改 users 表中的 userage 列，详细执行代码如下所示。

```
SQL> SET SERVEROUTPUT ON;
SQL> SELECT userage FROM users;
USERAGE
--------
20

SQL> UPDATE users SET userage 25;
```

```
修改前 20 修改后 25
已更新 1 行。
```

该代码首先查询 users 表中 userage 列值，然后执行 UPDATE 操作，将 userage 修改为 25。此时将会激活 trigger_user 触发器，获取修改前和修改后的值进行输出。

11.8.3 知识扩展——触发器的新值和旧值

在触发器中，可以使用 OLD 关键字来获取执行前的值，NEW 可以获取执行后的新值。在使用 OLD 关键字时只能在执行 UPDATE 和 DELETE 操作时才有效，而 NEW 关键字只能在执行 INSERT 和 UPDATE 语句时有效，例如下面实例中，创建一个名为 show_news 的触发器，该触发器中用于输出修改后的值，触发器代码如下。

```
SQL>CREATE OR REPLACE TRIGGER show_news
  2  BEFORE UPDATE ON users
  3  FOR EACH ROW
  4  DECLARE
  5  shownews VARCHAR;
  6  BEGIN
  7  shownews := :new.username;
  8  DBMS_OUTPUT.PUT_LINE('修改后的值：'||shownews);
  9  END;
 10  /
```

此时已经将触发器创建完成了，当用户修改 users 表中的数据时，系统将会把修改后的新数据输出到控制台中，例如下测试代码。

```
SQL> update users set username='小白' where id=20;
修改后的值：小白
已更新 1 行
```

11.8.4 触类旁通



使用 NEW 关键字未获取新值？

网络课堂：http://bbs.itzcn.com/thread-17120-1-1.html

学习了使用 NEW 和 OLD 关键字能够获取操作前后的数据，我就自己编写了一段代码如下，当执行 UPDATE 语句后获取 NEW 值，可是在执行 UPDATE 语句后控制台没有输出任何信息，什么原因。

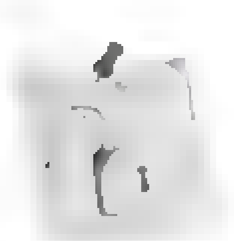
```
SQL> CREATE OR REPLACE TRIGGER delete_trigger_user
  2  BEFORE DELETE ON users
  3  FOR EACH ROW
  4  DECLARE
  5  newage NUMBER;
```

```
  6  BEGIN
  7  newage := :NEW.userage;
  8  DBMS_OUTPUT.PUT_LINE('删除后的值'||newage);
  9  END;
 10  /
触发器已创建

SQL> DELETE FROM users WHERE username='张三';
删除后的值
已删除 1 行。
```

在使用 NEW 关键字获取修改后的新值时它只针对 INSERT 和 UPDATE 语句有效。而针对 DELETE 没有作用，因为删除数据后就没有任何新数据的产生，所以也不会生成新的值。

11.8.5　网络课堂



视频教学：http://school.itzcn.com/video-vid-1218-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16839-1-1.html

11.9　如何将数据插入到复杂的视图中

11.9.1　问题描述

大家都知道，对视图的操作和对表的操作非常相似，下面代码为 users 表创建了视图并且将表中的信息添加到视图中。

```
SQL> CREATE OR REPLACE VIEW view_user
  2  AS
  3  SELECT username,userage+2 newage,usersex FROM users;
视图已创建。

SQL> SELECT * FROM view_user;
USERNAME          NEWAGE       USERSEX

张三              27           男
```

可是在使用 INSERT 语句为该视图添加数据时出现错误，代码如下所示。

```
SQL> INSERT INTO view_user VALUES('窗内网',8,'其他');
INSERT INTO view_user VALUES('窗内网',8,'其他')
*
第 1 行出现错误:
ORA-01733: 此处不允许虚拟列
```

11.9.2 解决方法

在操作视图时，Oracle 只能对简单的视图进行操作。如果视图中存在具有集合操作符、具有分组函数等等，在执行时就会出现错误异常。要想解决这样的问题，必须使用 INSTEAD OF 触发器，接下来创建一个名为 trigger_view_user 触发器。

```
SQL> CREATE OR REPLACE TRIGGER trigger_view_user
  2  INSTEAD OF INSERT ON view_user
  3  FOR EACH ROW
  4  BEGIN
  5  INSERT INTO users (username,userage,usersex) VALUES( :NEW.username,
  :NEW.us
erage, :NEW.usersex);
  6  END;
  7  /
触发器已创建
```

此时向 view_user 视图插入数据时，系统将会把新得到的数据插入到 users 表中。

11.9.3 知识扩展——INSTEAD OF 触发器

在 Oracle 数据库中，对于简单视图，可以直接执行 INSERT、UPDATE 和 DELETE 操作。如果视图中存在集合操作符、具有分组函数等将不可以直接进行 DML 操作。

可是在开发需求中可能不使用这个功能，因此在 PL/SQL 可以用建立 INSTEAD OF 触发器来完成这项任务。以下代码则是创建语法。

```
CREATE OR REPLACE TRIGGER trigger_name
INSTEAD OF INSERT ON view_name
FOR EACH ROW
BEGIN
   trigger_body
END;
```

语法代码中，trigger_name 表示创建的 INSTEAD OF 触发器名称；view_name 表示被操作的视图名称；

在创建 INSTEAD OF 触发器时还需要注意 INSTEAD OF 选项只适用于视图，并且不能指定 WITH CHECK OPTION、BEFORE 和 AFTER 选项，但是必须有 INSTEAD OF 选项。

11.9.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1220-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16840-1-1.html

第 12 章　用户权限与安全

Oracle 为了确保数据库系统的安全，为每一个用户账号都分配了一定的权限或者角色。所谓的权限就是针对每个用户设置不同的操作范围，这就是用户权限；角色就是一组或者一批相关权限的集合。也就是拥有该角色的用户将都会拥有相同的权限。这样系统管理员能够更加方便地操作用户权限。很多初学者在为用户账号指定权限时，往往会出现这样那样的错误，本章将这些问题集中起来进行讲解。

12.1　如何创建 Oracle 用户

12.1.1　问题描述

Oracle 用户对于 Oracle 数据库来说就相当于一把开门的钥匙，如果没有这个钥匙是无法进入 Oracle 系统的。那么如何才能获得这把钥匙，怎样为 Oracle 创建新的用户？

12.1.2　解决方法

在 Oracle 数据库中支持管理员创建新用户的功能，不过前提是该管理员必须具备创建用户的 CREATE USER 权限。下面代码是使用 SYSTEM 登录系统后，创建用户名为 test，口令为 test123 的用户。

```
SQL> CREATE USER test
  2  IDENTIFIED BY test123
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp
  5  QUOTA 20M ON users;
用户已创建。
```

上述代码成功地创建了 test 用户，并且设置了该用户所属默认表空间 USERS 和临时表空间 TEMP 等信息。

12.1.3　知识扩展——用户的概念

用户和数据库之间的关系就像钥匙和锁的关系。只有拥有这把钥匙才可以对门后面的事务进行操作。同样在 Oracle 数据库中只有用户通过登录后，才可以对数据执行一些数据和其他操作。

由于每个数据库用户的权限或者角色不同，所以他们具备的能力也不相同。但是对于数据库来说，每个数据库都将会配备一个数据库管理员，对其数据进行管理操作。数据库管理员的主要任务如下：安装和升级 Oracle 数据库服务、建立数据库、建立数据库的主要存储结构（表空间）、建立数据库的主要对象（表、视图、索引）和制定并实施备份与恢复计划。

根据 DBA 承担的管理职责不同，可以为他们分配不同的数据库用户，管理数据库的用户主要包括特权用户和 DBA 用户两种类型：

1．特权用户

权限用户拥有一些特殊的功能权限，例如 SYSDBA 或者 SYSOPRE 等权限用户，而这些用户可以启动例程、关闭例程、执行备份和恢复等操作。在 Oracle 11g 中，Oracle 提供了默认的特权用户 SYS，当以特权用户身份登录数据时，必须带有 AS SYSDBA 或 AS SYSOPER 选项。例如：

```
SQL> CONNECT SYS/123 AS SYSDBA
已连接。
```

2．DBA 用户

DBA 用户是指具有 DBA 角色的数据库用户。特权用户可以启动例程（实例）、关闭例程等特殊操作，而 DBA 用户只有在启动了数据库之后才能执行各种管理操作。默认的 DBA 用户为 SYS 和 SYSTEM。



SYS 用户不仅具有 SYSDBA 和 SYSOPER 特权，而且还具有 DBA 角色，因此它不仅可以启动例程、停止例程，也可以执行任何管理操作。而 SYSTEM 用户只具有 DBA 角色，因此不能启动例程、停止例程。

12.1.4 知识扩展——创建用户的语法

创建用户就相当于为这把锁又配置了一把钥匙。在创建数据库用户时需要使用 CREATE USER 语句来完成。但是要想操作该语句，系统默认只有 DBA 用户才可以。如果其他用户要想操作该语句必须拥有 CREATE USER 系统权限才可以。创建语法如下所示。

```
CREATE USER user name
IDENTIFIED BY password
[ DEFAULT TABLESPACE default tablespace |
TEMPORARY TABLESPACE temp tablespace |
PROFILE profile
QUOTA [ integer K | M ] | UNLIMITED ON tablespace
| PASSWORD EXPIRE
| ACCOUNT LOCK | UNLOCK ];
```

关于 CREATE USER 语句的各个字句的功能简介如下：

- **user_name**

user_name 表示创建的数据库用户登录名。

- **password**

password 表示登录口令。

- **default_tablespace**

default tablespace 表示为用户指定默认表空间，如果用户没有指定该表空间，则系统将使用用户默认的表空间来存储。

- **temp_tablespace**

temp tablespace 表示为该用户设置临时表空间，如果没有指定用户临时表空间，则系统将使用默认的临时表空间来存储。

- **profile**

profile 表示用户的资源文件，该资源文件必须在之前已经被创建。在安装数据库时，Oracle 自动建立名为 DEFAULT 的默认资源文件。如果没有为用户指定 PROFILE 选项，则 Oracle 将会为它指定 DEFAULT 资源文件；

- **PASSWORD EXPIRE**

PASSWORD EXPIRE 表示将用户口令的初始状态设置为已过期，从而强制用户在每一次登录数据库后必须修改口令。

- **ACCOUNT LOCK|UNLOCK**

ACCOUNT LOCK | UNLOCK 设置用户的初始状态为锁定（LOCK）或解锁（UNLOCK），表示锁定或者解锁某个用户账号。

12.1.5 触类旁通



使用语句创建用户出错？

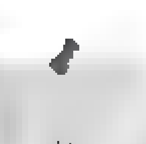
网络课堂：http://bbs.itzcn.com/thread-16842-1-1.html

使用 CREATE USER 语句创建了 test 用户之后，无法登录，提示：用户名和口令无效。可是我这次重新创建该用户时却出现了这样的错误，代码如下所示。

```
SQL> CREATE USER test
  2  IDENTIFIED BY test111
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp
  5  QUOTA 20M ON users;
CREATE USER test
            *
第 1 行出现错误:
ORA-01920: 用户名 'TEST' 与另外一个用户名或角色名发生冲突
```

从错误提示上看，提示 test 用户已经存在，所以第一次创建的 test 用户已经成功，用户名已经存在或者系统在你没有创建之前就已经存在了该用户，因此造成用户名或角色名冲突。可以删除该用户的相关信息，然后再进行创建操作。

12.1.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-1222-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16841-1-1.html

12.2 新建用户为什么会失效

12.2.1 问题描述

在 Oracle 数据库中，使用 SQL 语句为数据库添加了一个新用户，可是新添加的用户无法正常登录，详细代码如下所示。

```
SQL> CREATE USER teacher
  2  IDENTIFIED BY 123456
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp
  5  QUOTA 30M ON users
  6  PASSWORD EXPIRE;
用户已创建。
请输入用户名:  teacher
输入口令:  12346
ERROR:
ORA-28001: the password has expired
更改 teacher 的口令
新口令:
```

上述代码中，使用 CREATE USER 语句创建了名为 teacher 的用户，可是在登录时却出现了重新设置新密码的提示信息，为什么新创建的用户失效了？

12.2.2 解决方法

因为在创建表时，设置了 PASSWORD EXPIRE 属性，它是用来设置用户口令失效，设置之后强制用户在登录数据库时必须修改口令。不过新创建的用户没有权限修改自己的密码，需要使用超级管理员登录后给该用户添加修改密码的权限。代码如下所示。

```
SQL> CONNECT SYSTEM/tiger AS SYSDBA
已连接。
SQL> GRANT CONNECT,RESOURCE TO teacher;
授权成功。
```

权限赋予之后，用户使用 teacher 用户对其进行密码重置，然后就可以成功登录到 Oracle 系统中。

12.2.3 知识扩展——使用 PASSWORD EXPIRE 语句

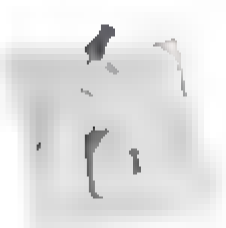
该语句主要用来设置用户口令过期、失效，强制用户在登录数据库时必须修改口令。其

语法如下：

```
SQL> CREATE USER dsz
  2  IDENTIFIED BY 123
  3  DEFAULT TABLESPACE users
  4  PASSWORD EXPIRE;
用户已创建。
```

上述代码是 PASSWORD EXPIRE 语句的使用方法，这样新创建的用户在登录时就需要重新设置登录密码了。

12.2.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-3974-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16843-1-1.html

12.3 error：the account is locked 是什么原因

12.3.1 问题描述

前面学习了在创建用户时可以设置用户的状态为锁定，可是设置的状态为锁定后，就无法正常地登录到 Oracle 系统中，例如以下代码则是使用被锁定用户进行登录的异常信息，如何为该用户进行解锁让该用户可以登录到系统中？

```
请输入用户名： teacher/123456
ERROR:
ORA-28000: the account is locked
```

12.3.2 解决方法

解锁用户有两种方法，一种是在管理员创建用户时设置该用户的状态为解锁状态，另外一种是对已经存在的用户进行解锁操作。创建用户是进行解锁设置和创建用户进行加锁设置一样，只需要为该用户设置 ACCOUNT UNLOCK 的选项即可。那么要对已经存在的用户解锁代码如下所示。

```
SQL> ALTER USER teacher ACCOUNT UNLOCK;
用户已更改。
```

上述代码则是为已经存在的 teacher 用户进行解锁，但是要注意的是，要对用户进行解锁操作当前登录的用户必须拥有该权限才可以进行操作。

12.3.3 知识扩展——使用 ACCOUNT LOCK 或者 ACCOUNT UNLOCK 选项

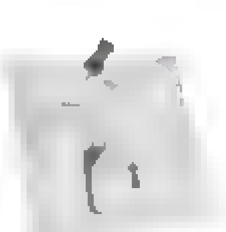
在创建新用户时，不仅可以对用户密码进行失效，还可以对该用户进行锁定和解锁等操作，需要使用 ACCOUNT LOCK 或者 ACCOUNT UNLOCK 选项。ACCOUNT LOCK 表示锁定用户；ACCOUNT UNLOCK 表示解锁用户，默认情况下是解锁状态。

例如以下代码为新创建的 teacher 用户进行加锁操作。

```
SQL> CREATE USER teacher
  2  IDENTIFIED BY 123456
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp
  5  QUOTA 20M ON USERS
  6  ACCOUNT LOCK;
用户已创建。
```

代码中新创建了 teacher 用户，并且设置该用户密码为 123456。此时用户的状态为锁定状态。

12.3.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-3975-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16844-1-1.html

12.4 数据库用户密码被泄露，怎么办

12.4.1 问题描述

当我在登录 Oracle 数据库系统时，提示密码输入错误，可是我明明记得设置的登录密码没错，为什么会出现这种现象呢？是不是我的密码被盗了？是否可以在 Oracle 数据库中修改用户密码呢？怎么操作？

12.4.2 解决方法

如果在使用 Oracle 数据用户登录系统时，发现口令泄露，需要修改口令可以使用 ALTER USER…IDENTIFIED BY 语句进行完成。例如以下代码是修改 teacher 用户的口令信息。

```
SQL>ALTER USER teacher IDENTIFIED BY 111111;
用户已更改。
```

上述代码中，USER 右侧是需要修改口令的用户名，BY 右侧则是要修改的口令，该口令可以是任何字符串。

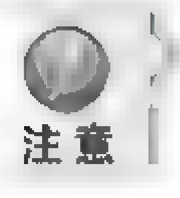
Oracle 除此之外修改口令的方法，还有使用 GRANT 命令修改用户的口令，例如以下代码同样修改 teacher 用户口令。

```
SQL> GRANT CONNECT TO teacher IDENTIFIED BY 000000;
授权成功。
```

12.4.3 知识扩展——修改用户

对于 Oracle 管理员而言，重新修改账户信息是经常需要执行的一种操作。为了方便对用户信息的操作，Oracle 提供了一套修改用户信息的语句，即 ALTER USER 语句，该语句的语法如下：

```
ALTER USER user_name
IDENTIFIED BY password
[ DEFAULT TABLESPACE tablespace |
TEMPORARY TABLESPACE tablespace |
PROFILE profile
QUOTA [ integer K | M ]
| UNLIMITED ON tablespace
| PASSWORD EXPIRE
| ACCOUNT LOCK | UNLOCK ];
```


注意

在修改用户信息时，用户的名称是不可以进行修改的。如果需要强制修改用户名，那么建议删除该用户重新创建新用户名称。

1. 修改口令

修改口令的操作在数据库管理方面是经常被使用到的，在执行修改口令时，操作用户必须具备相应的修改权限或者是管理员才可以进行修改操作。下面则是使用 ALTER USER… IDENTIFIED BY 语句修改口令的语法结构。

```
ALTER USER user_name IDENTIFIED BY new_password
```

在该语法中，user name 表示被修改口令的用户名称；new password 表示要修改的新口令，可以为任何字符串。

GRANT 命令修改用户的口令语法如下所示。

```
GRANT CONNECT TO user_name IDENTIFIED BY new_password
```


提示

对某个用户的账号进行修改，不会立即应用到该用户的当前会话中，而是在该用户下一次连接数据库时生效。

2．修改默认表空间

对用户默认表空间进行修改，需要使用 ALTER USER … DEFAULT TABLESPACE 语句。其语法如下所示。

```
ALTER USER user_name DEFAULT TABLESPACE tablespace_name
```

该语法中，user_name 表示要被修改的用户名称；tablespace_name 表示修改后的新默认表空间名称。


注意

对用户默认表空间修改后，先前已经创建的表仍然存储在原表空间中。如果创建新表，则将存储在新表空间中。

3．修改临时表空间

对用户临时表空间进行修改，需要使用 ALTER USER … TEMPORARY TABLESPACE 语句。语法如下所示。

```
ALTER USER user_name TEMPORARY TABLESPACE tablespace_name
```

该语法中，user_name 表示被修改的用户名称；tablespace_name 被修改后的临时表空间名称。

4．修改表空间的配额

对用户的表空间配额进行修改，需要使用 ALTER USER … QUOTA 语句。修改语法如下所示。

```
ALTER USER user_name QUOTA size ON SYSTEM;
```

12.4.4　网络课堂



视频教学：http://school.itzcn.com/video-vid-1223-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16845-1-1.html

12.5　删除用户时提示 ORA-01940 的错误信息

12.5.1　问题描述

对用户的删除操作也是经常会被使用到的，对于不需要的用户可以进行删除，可是在执行删除该用户时却出现了异常错误，删除语句和错误信息如下所示。

```
SQL> CONNECT teacher/000000;
已连接。
SQL> DROP USER teacher;
```

```
drop user teacher cascade
*
ERROR at line 1;
ORA-01940: cannot drop a user that is currently connected
```

在执行删除 teacher 用户时，出现了 cannot drop a user that is currently connected 错误，如何解决？

12.5.2 解决方法

如果被删除的用户正在连接数据库，则必须在该用户退出系统后才能完成删除，否则，在删除时将提示错误信息。

你可以选择使用其他用户或者 SYSTEM 用户登录。然后删除 teacher 用户，代码如下所示。

```
SQL> CONNECT system/tiger;
已连接。
SQL>DROP USER teacher;
用户已删除。
```

12.5.3 知识扩展——删除用户

删除用户时，需要使用 DROP USER 语句，该语句的语法如下：

```
DROP USER user_name [ CASCADE ];
```

该语法中，user_name 表示将要被删除的用户名称；CASCADE 选项表示在删除用户时，将该用户所创建的模式对象也全部删除。

例如在下面实例代码中，通过使用 DROP 语句对 teacher 用户进行删除，其删除代码如下。

```
SQL> DROP USER teacher
用户已删除。
```

12.5.4 触类旁通



删除用户又出现 ORA-01992 问题？

网络课堂：http://bbs.itzcn.com/thread-16847-1-1.html

解决了上面的问题之后，第一个用户是删除掉了。可是在删除其他需要删除的用户时却出现了新问题，删除代码如下所示。

```
SQL> DROP USER dsz;
drop user dsz
*
ERROR at line 1;
ORA 01992: CASCADE must be specified to drop 'DSZ'
```

如果用户已经创建了模式对象，则删除用户时必须使用 CASCADE 选项，否则系统将提示错误信息。正确的删除语法如下所示。

```
SQL>DROP USER dsz CASCADE;
用户已删除。
```

12.5.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-3976-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16846-1-1.html

12.6 如何终止用户当前的会话

12.6.1 问题描述

今天在管理数据库时，为了确保数据库的安全性就随意地查看了一下用户登录监控系统，发现有一个非内部人员的用户正在登录该系统并且进行一些操作，如何才可以终止该用户的会话？各位帮个忙解答下事关重要，快点啦！

12.6.2 解决方法

要获取用户的登录信息，可以查询数据字典视图 V$SESSION。在该数据字典中 SID 和 SERIAL#列能够表示一个会话，因此要终止某一个会话可以使用这两列作为条件能够确认要终止的会话。

```
SQL> ALTER SYSTEM KILL SESSION '25,1';
系统已更改。
```

12.6.3 知识扩展——查看用户会话信息

当用户连接数据库后，会创建一个会话。其中每个用户都将会拥有一个会话。此时数据库管理员可以查看系统提供的字典视图，根据会话情况了解用户的操作，在必要时通过监控与限制的方式，防止用户无限制地使用系统资源。

1. 使用数据字典视图 V$SESSION

通过查询数据字典视图 V$SESSION 可以获取用户的会话信息，例如用户登录时间、会话号和计算机名等。

```
SQL> SELECT SERIAL#,USERNAME, LOGON TIME, MACHINE
  2  FROM V$SESSION WHERE username IS NOT NULL;
```

```
SERIAL#     USERNAME         LOGON TIME       MACHINE
--------    ----------       ------------     ------------
1                            25-7月 -11       DSZ
1                            25-7月 -11       DSZ
......
26                           25-7月 -11       DSZ
2           SYSTEM           25-7月 -11       WORKGROUP\ DSZ
已选择 22 行。
```

上述代码中，LOGON_TIME 表示该用户登录数据库的时间；USERNAME 表示用户名；MACHINE 表示用户登录数据库时所使用的计算机名。

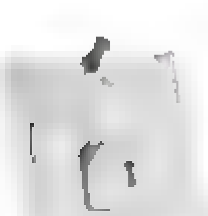
2．使用数据字典视图 V$OPEN_CURSOR

在数据字典视图 V$OPEN_CURSOR 中，记录了用户连接数据库后所执行的 SQL 语句。以下代码则是查询 V$OPEN_CURSOR 字典视图的结果信息。

```
SQL> SELECT SID,SQL_TEXT,USER_NAME
  2  FROM V$OPEN_CURSOR WHERE USER_NAME='SYSTEM';

SID     SQL_TEXT                                                USER_NAME
------ -------------------------------------------------------- ----------
125     BEGIN DBMS_APPLICATION_INFO.SET_MODULE(:1,NULL); END;  SYSTEM
125     SELECT DECODE('A','A','1','2') FROM DUAL                SYSTEM
125     BEGIN DBMS_OUTPUT.DISABLE; END;                         SYSTEM
......
已选择 6 行。
```

12.6.4　网络课堂



视频教学：http://school.itzcn.com/video-vid-1224-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16848-1-1.html

12.7　为什么新创建的用户无法创建其他用户

12.7.1　问题描述

今天遇到一个比较头疼的问题。使用新创建的用户名和口令登录到系统中之后，无法创建其他用户，我还以为是自己创建用户时出的错，将该用户删除并重新创建。可是使用新创建的用户就是无法成功地创建其他用户。

```
SQL> CONNECT teacher/123456
已连接。
```

```
SQL> CREATE USER dsz
  2  IDENTIFIED BY 123456
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp
  5  ACCOUNT LOCK;
identified by 123456
              *
第 2 行出现错误:
ORA-01031: 权限不足
```

当使用新创建的 teacher 用户创建其他用户时，出现权限不足的问题。这是什么原因造成的。

12.7.2 解决方法

因为新的用户没有创建其他用户的权限，只有在管理员为该用户赋予创建用户的权限之后，才可以使用该用户创建其他用户。以下代码就是为新用户赋予创建用户的权限，并且使用赋予好权限的用户创建其他用户。

```
SQL> CONNECT system/tiger          --切换管理员登录
已连接。
SQL> GRANT CREATE USER TO teacher; --赋予创建用户权限
授权成功。
SQL> connect teacher/123456        --切换 teacher 用户
已连接。
SQL> CREATE USER dsz               --创建用户
  2  IDENTIFIED BY 123456
  3  DEFAULT TABLESPACE users
  4  TEMPORARY TABLESPACE temp
  5  ACCOUNT LOCK;
用户已创建。
```

上述代码中，首先切换到 system 用户登录，为 teacher 用户赋予创建用户的权限。然后再切换至 teacher 用户登录，执行创建用户代码。此时用户即可创建成功。

12.7.3 知识扩展——系统权限

系统权限的作用就是用于控制数据的操作机制，这些权限是 Oracle 定义在系统中，且无法进行修改和扩展。系统权限包括连接数据库、创建表空间、创建会话、创建表等等操作权限。如果用户没有这些权限是无法进行这样的操作的。

1．Oracle 常用系统权限

Oracle 系统权限又分很多种，其中每个权限都有各自的责任，只有用户拥有相关权限才可以进行相关操作。Oracle 数据字典视图 SYSTEM PRIVILEGE MAP 中保存着所有的系统

权限，在表 12-1 中就是 Oracle 常用系统权限。

表 12-1　Oracle 常用系统权限

系统权限	说明
CREATE SESSION	连接数据库
CREATE TABLESPACE	创建表空间
ALTER TABLESPACE	修改表空间
DROP TABLESPACE	删除表空间
CREATE USER	创建用户
ALTER USER	修改用户
DROP USER	删除用户
CREATE TABLE	创建表
CREATE ANY TABLE	在任何用户模式中创建表
DROP ANY TABLE	删除任何用户模式中的表
ALTER ANY TABLE	修改任何用户模式中的表
SELECT ANY TABLE	查询任何用户模式中基本表的记录
INSERT ANY TABLE	向任何用户模式中的表插入记录
UPDATE ANY TABLE	修改任何用户模式中的表的记录
DELETE ANY TABLE	删除任何用户模式中的表的记录
CREATE VIEW	创建视图
CREATE ANY VIEW	在任何用户模式中创建视图
DROP ANY VIEW	删除任何用户模式中的视图
CREATE ROLE	创建角色
ALTER ANY ROLE	修改任何角色
GRANT ANY ROLE	将任何角色授予其他用户
ALTER DATABASE	修改数据库结构
CREATE PROCEDURE	创建存储过程
CREATE ANY PROCEDURE	在任何用户模式中创建存储过程
ALTER ANY PROCEDURE	修改任何用户模式中的存储过程
DROP ANY RPOCEDURE	删除任何用户模式中的存储过程
CREATE PROFILE	创建配置文件
ALTER PROFILE	修改配置文件
DROP PROFILE	删除配置文件

2. 系统权限设置

权限设置也是对用户进行功能的发放，通常这些任务都是由数据库管理员来完成的。因为普通的用户不具备分配权限的资格。要想让普通用户具有分配权限的资格，那么该用户必须拥有 GRANT ANY PRIVILEGE 系统权限，以下代码则是设置权限的语法结构。

```
GRANT SYSTEM PRIV [,SYSTEM PRIV,…]
TO {PUBLIC |role | user}[,{user | role | PUBLIC}]…
[WITH ADMIN OPTION];
```

其中：

- **SYSTEM_PRIV**

表示设置的权限名称，如果有多个则用逗号分开。

- **user**

表示被设置权限的用户名。

- **role**

表示被设置的角色名称。

- **WITH ADMIN OPTION**

表示拥有将相应的系统权限授予其他用户、角色。

3．回收系统权限

回收系统权限和设置系统权限的过程是相反的。不过设置和回收系统权限都是由数据库管理员来完成的，普通用户只有在拥有 WITH ADMIN OPTION 权限才可以进行设置。通常回收系统权限就是删除该用户的某些系统权限，需要使用 REVOKE 语句完成，语法结构如下：

```
REVOKE SYSTEM PRIV[ , SYSTEM PRIV] …
FROM PUBLIC | role | user_name
```

该语法中，SYSTEM_PRIV 表示要收回的系统权限名称；PUBLIC 就表示要收回系统权限的用户名称或者是角色名称。

12.7.4 触类旁通



有权限创建用户，为什么没有权限回收用户权限？

网络课堂：http://bbs.itzcn.com/thread-16851-1-1.html

在 Oracle 系统中，有一个名为 teacher 的用户。使用该用户可以创建新用户信息，确实在设置或者回收其他用户权限时出现异常，异常代码如下所示。

```
SQL> CONNECT teacher/123456
已连接。
SQL> REVOKE CREATE TABLE FROM dsz;
REVOKE CREATE TABLE FROM dsz
*
第 1 行出现错误:
ORA-01031: 权限不足
```

上述代码是使用 teacher 用户登录 Oracle 系统中，可是在执行 REVOKE 语句时出现了权限不足的异常信息，这是什么原因？

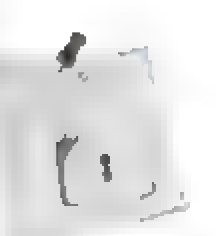
普通用户要对其他用户执行授予或者去除系统权限，那么该用户必须拥有 WITH ADMIN OPTION 权限。上述问题可以这样操作：首先使用 system 用户登录系统，为 teacher 设置该权限，然后再切换至 teacher 用户进行权限的操作。正确代码如下：

```
SQL> CONNECT system/tiger
已连接。
```

```
SQL> GRANT CREATE TABLE TO teacher WITH ADMIN OPTION;
授权成功。
SQL> CONNECT teacher/123456
已连接。
SQL> GRANT CREATE TABLE TO dsz;
授权成功。
```

在上述代码中，首先使用 system 用户登录后为 teacher 用户设置了 CREATE TABLE 权限，并且设置该用户 WITH ADMIN OPTION 权限。此时 teacher 用户可以将 CREATE TABLE 权限设置为其他用户。

12.7.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1228-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1229-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-16850-1-1.html

12.8 无法将数据插入数据库表中

12.8.1 问题描述

我新建了一个具有创建表权限的 teacher 用户。并利用该用户创建了 userinfo 表。可是在该表中插入数据出现了权限问题，是不是该用户没有插入数据和删除数据的权限呢？如何为 teacher 表设置相关权限？

以下代码则是使用 teacher 用户登录后，为 userinfo 表添加数据时出现的异常信息。

```
SQL> INSERT INTO userinfo VALUES('dsz','123456',22);
INSERT INTO userinfo VALUES('dsz','123456',22)
            *
第 1 行出现错误:
```

12.8.2 解决方法

根据你的描述发现，teacher 只拥有创建表的权限是不够的，需要为用户添加对象权限。所谓的对象权限就是对表、视图、目录等对象进行操作的权限。在插入数据时出现这样的错误其原因在于该用户没有表的操作权限。可以为该用户设置 TABLE 对象权限，详细代码如下所示。

```
SQL> connect system/tiger
已连接。
```

```
SQL> GRANT ALL ON teacher.userinfo TO teacher;
授权成功。
SQL> connect teacher/123456
已连接。
SQL> INSERT INTO userinfo VALUES('dsz','123456',22);
已创建 1 行。
```

上述代码为 teacher 用户赋予对 TABLE 对象操作权限之后，使用 teacher 用户就可以将数据插入到 userinfo 表中了。

12.8.3 知识扩展——对象权限

相对于系统权限，对象权限是在 Oracle 数据库对象上设置类型权限，该对象权限不在用户自己的模式中。操作权限时需要使用 GRANT 和 REVOKE 命令。

1. 常用对象权限

常用的 Oracle 数据库对象有 7 个，分别为：表、视图、目录、函数、存储过程、包或者序列等对象。而对这些对象有 9 种操作权限，表 12-2 列出了对象权限的分类。

表 12-2 对象权限类型

权限 / 对象	ALTER	DELETE	EXECUTE	INDEX	INSERT	RREAD	REFERENCE	SELECT	UPDATE
TABLE	YES	YES		YES	YES		YES	YES	YES
VIEW		YES			YES			YES	YES
DIRECTORY						YES			
FUNCTION			YES						
PROCEDURE			YES						
PACKAGE			YES						
SEQUENCE	YES							YES	

在上表中，凡是在对应的表格中拥有 YES 标识，说明该对象拥有对应权限。如果对应表格中没有任何内容就表示当前对象不具备这样的权限。

2. 对象权限设置

设置对象权限的方法和设置系统权限的方法相同，需要使用 GRANT 语句来完成。在设置权限时，只有对象的拥有者才能为其他用户设置权限，非对象的拥有者不可以向其他用户设置权限。其语法如下：

```
GRANT object privilege | ALL [PRIVILEGES]
 ON [schema.]object
 TO PUBLIC | role | user name
 [WITH GRANT OPTION] ;
```

该语法中，object_privilege 表示对象权限。设置是将对象权限名称设置在这里即可；ALL [PRIVILEGES]表示所有对象权限，不过必须是当前对象所具备的权限；ON 后面表示用户模

式及对象名称。TO 后面则是被设置对象权限的用户名称；WITH GRANT OPTION 表示允许用户将该对象权限授予其他用户。与授予系统权限的 WITH ADMIN OPTION 子句相类似。

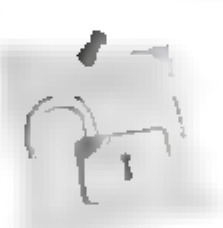
3．回收对象权限

回收对象权限通常是由对象拥有者来完成的任务。如果其他用户必须拥有回收的权限。回收对象权限和回收系统权限也非常相似，都是通过使用 REVOKE 语句进行回收操作，语法如下：

```
REVOKE object privilege | ALL [PRIVILEGES]
 ON [ schema.]object
 FROM PUBLIC | role | user;
```

在该语法中，object_privilege 表示要被回收的对象权限；ALL 表示回收所有的对象权限；ON 后面表示回收权限的模式和对象名称；FROM 后是针对回收的用户名称。

12.8.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1230-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1231-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16852-1-1.html

12.9 如何为多个用户设置或者取消相同的权限

12.9.1 问题描述

在实际的数据库管理中，经常会为多个用户赋予相同的权限。如果为每个用户手动设置这些相同的权限，可能会浪费没必要的精力和时间。是否有好的办法为多个用户进行相同权限的管理？

12.9.2 解决方法

为了方便对具有相同权限的用户进行权限的统一管理，Oracle 设置了角色功能。Oracle 管理员可以创建一个角色，然后为该角色设置相应的权限信息，最后将该角色授予某个用户，这样该用户就拥有当前角色的所有权限。

例如以下代码中，创建一个 school 角色，然后为角色设置权限，最后将该角色赋予 teacher 用户。

```
SQL> CONNECT system/tiger
已连接。
SQL> CREATE ROLE school IDENTIFIED BY PASSWORD;
角色已创建。
```

```
SQL> GRANT CREATE SESSION,CREATE TABLE,CREATE VIEW TO school WITH ADMIN OPTION;
授权成功。
SQL> GRANT school TO teacher;
授权成功。
SQL> CONNECT teacher/123456
已连接。
```

上述代码中，创建了 school 角色，并且为该角色添加了登录、创建表和创建视图的权限。然后将该角色赋予到 teacher 用户中，那么此时 teacher 用户就拥有了这 3 个权限。

该角色可以为多个用户进行授予，这样就可以对相同权限用户进行权限的统一管理。

12.9.3 知识扩展——系统角色

众所周知，在 Oracle 系统中权限比较多，管理起来比较麻烦。为了能够更好地管理这些权限，可以使用角色将一组权限封装起来统一管理。而在 Oracle 系统中默认一个特殊的角色，并且为这些角色授予了相应的权限。关于这些角色的简介如下。

1. CONNECT 角色

CONNEC 角色是在创建数据库时系统自动生成的角色。在该角色中拥有开发人员所需的多数权限，是授予最终用户的典型权利。表 12-3 则是该角色的权限。

表 12-3 CONNECT 角色权限

权限名称	说明
ALTER SESSION	建立会话
CREATE CLUSTER	建立聚簇
CREATE DATABASE LINK	建立数据库链接
CREATE SEQUENCE	建立序列
CREATE SESSION	建立会话
CREATE SYNONYM	建立同义词
CREATE VIEW	建立视图
CREATE TABLE	建立表

2. RESOURCE 角色

RESOURCE 角色也是在创建数据库时，Oracle 执行脚本 SQL.BSQ 自动建立的角色。在该角色中存储着如表 12-4 所示的权限。

表示 12-4 RESOURCE 角色权限

权限名称	说明
CREATE CLUSTER	建立聚簇
CREATE PROCEDURE	建立过程
CREATE SEQUENCE	建立序列
CREATE TABLE	建立表
CREATE TRIGGER	建立触发器
CREATE TYPE	建立类型

3．DBA 角色

DBA 角色是在创建数据库时创建的角色对象。该角色拥有所有的系统权限和 WITH ADMIN OPTION 选项。凡是赋予该角色的用户都将是 Oracle 数据库的超级管理员，例如默认的 SYSTEM 或者 SYS 等用户都拥有 DBA 角色。

4．EXP_FULL_DATABASE 角色

EXP_FULL_DATABASE 角色是安装数据字典时系统自动生成的角色。该角色拥有数据库导出权限，详细如表 12-5 所示。

表 12-5　EXP_FULL_DATABASE 角色权限

权限名称	说　明
BACKUP ANY TABLE	备份任何表
EXECUTE ANY PROCEDURE	执行任何过程、函数和包
SELECT ANY TABLE	查询任何表
EXECUTE ANY TYPE	执行任何对象类型
ADMINISTER_RESOURCE_MANAGER	管理资源管理器
EXECUTE_CATALOG_ROLE	执行任何 PL/SQL 系统包
SELECT_CATALOG_ROLE	查询任何数据字典

5．IMP_FULL_DATABASE 角色

IMP_FULL_DATABASE 角色用于执行数据库导入操作，它包含了 EXECUTE_CATALOG_ROLE、SELECT_CATALOG_ROLE 角色和大量的系统权限。

6．RECOVERY_CATALOG_OWNER 角色

RECOVERY_CATALOG_OWNER 角色为恢复目录所有者提供了系统权限，详细权限如表 12-6 所示。

表 12-6　RECOVERY_CATALOG_OWNER 角色权限

权限名称	说　明
CREATE SESSION	建立会话
ALTER SESSION	修改会话参数设置
CREATE SYNONYM	建立同义词
CREATE VIEW	建立视图
CREATE DATABASE LINK	建立数据库链接
CREATE TABLE	建立表
CREATE CLUSTER	建立簇
CREATE SEQUENCE	建立序列
CREATE TRIGGER	建立触发器
CREATE PROCEDURE	建立过程、函数和包

12.9.4　知识扩展——角色操作

角色就是一组权限的集合。在角色中拥有着不同的权限，对角色的管理就是对权限的管理。在 Oracle 系统中，不仅有系统自带的角色，还可以自定义适合自己的角色信息。

1．创建角色

使用 SQL 语句创建角色时需要使用 CREATE ROLE 语句来完成，其前提是创建者必须拥有 CREATE ROLE 权限。CREATE ROLE 语法如下：

```
CREATE ROLE role name
 [ NOT IDENTIFIED | IDENTIFIED BY password ];
```

该语法中 role name 表示创建的角色名称；password 表示为角色口令，默认为 NOT IDENTIFIED。

在 Oracle 系统中，为具有相同权限的用户进行分组，这样就称为角色。如果需要对新的用户添加相同的权限时，只需要为该用户赋予角色即可。

2．禁用和启用角色

禁用和启用角色也是经常会被使用到的功能。当某个角色由于某种原因需要停止该角色的所有权限时，就需要通过使用 SET ROLE 语句禁用该角色。其语法如下：

```
SET ROLE [ role name [ IDENTIFIED BY password ]
 | role name [ IDENTIFIED BY password ] … ]
 | ALL [ EXCEPT role_name[,role_name] ]
 | NONE
];
```

其中，role_name 表示被操作的角色名称；password 表示角色密码；ALL 关键字表示将启用用户被赋予的所有角色；EXCEPT role_name 表示除指定的角色外，启用其他全部角色；NONE 表示禁用指定的角色。

3．修改角色

如果在成功创建角色之后发现角色口令信息需要进行修改，可是使用 ALTER ROLE 语句可以对角色的口令进行修改，语法如下：

```
ALTER ROLE role name
[ NOT IDENTIFIED |
IDENTIFIED BY password
];
```

其中，role_name 表示修改的角色名称；NOT IDENTIFIED 表示不需要口令就可以启用或修改该角色；IDENTIFIED BY password 表示必须通过指定口令才能启用或修改该角色。

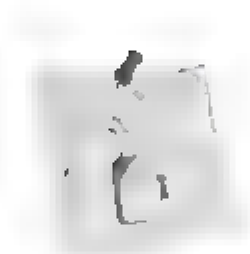
4．删除角色

如果需要删除当前数据库中的某个角色，可以使用 DROP ROLE 语句执行操作。删除语法如下。

```
DROP ROLE role_name;
```

删除角色时，该角色所拥有的权限也将会消失，并且拥有该角色的所有用户也将会失去相应的权限功能。

12.9.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1232-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1233-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1234-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16853-1-1.html

12.10 如何查看用户的所有角色

12.10.1 问题描述

Oracle 管理员为 teacher 用户添加了 school 角色，并且该角色拥有一些权限，可是后来忘记了添加的角色和相应的权限，因此在创建表时出现权限错误提示。是否可以查看用户所属的角色和相应的权限来确保该用户拥有创建表的权限。

12.10.2 解决方法

使用 system 用户登录 Oracle 系统，然后查询 DBA_ROLE_PRIVS 数据字典视图数据。在该字典视图数据中存储着所有用户对应的角色信息，例如以下代码则是查询 teacher 用户所拥有的角色信息。

```
SQL> CONNECT system/tiger
已连接。
SQL> SELECT grantee,granted_role FROM DBA_ROLE_PRIVS
  2  WHERE grantee='TEACHER';
GRANTEE                        GRANTED ROLE
----------------------         ---------------------------
TEACHER                        SCHOOL
```

在查询结果中 GRANTED_ROLE 列表示所拥有的角色信息。如果需要查看该角色所拥有的权限信息可以查询数据字典视图 ROLE_SYS_PRIVS 中的数据信息，代码如下所示。

```
SQL> SELECT * FROM ROLE SYS PRIVS WHERE role='SCHOOL';
ROLE                PRIVILEGE                ADM
------------------- ------------------------ ---------------
SCHOOL              CREATE VIEW              YES
SCHOOL              CREATE SESSION           YES
SCHOOL              CREATE TABLE             YES
```

这样，teacher 用户就拥有了 CREATE VIEW、CREATE SESSION 和 CREATE TABLE 三个权限。

12.10.3 知识扩展——查看角色信息

在 Oracle 系统中，如果要查看有关角色或者权限信息，可以查询相应的数据字典视图，通过查询这些字典视图中的数据可以获取相应的信息。表 12-7 是 Oracle 中存储角色信息的数据字典视图。

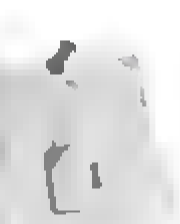
表 12-7 角色字典视图

视图	说明
DBA_ROLES	记录数据库中所有的角色
DBA_ROLE_PRIVS	记录所有已经被授予用户和角色的角色
USER_ROLES	包含已经授予当前用户的角色信息
ROLE_ROLE_PRIVS	包含角色授予的角色信息
ROLE_SYS_PRIVS	包含为角色授予的系统权限信息
ROLE_TAB_PRIVS	包含为角色授予的对象权限信息
SESSION_ROLES	包含当前会话所包含的角色信息

例如下面实例代码中，通过查询角色字典来获取当前正在回话的所有角色信息，查询 SESSION_ROLES 视图代码。

```
SQL> SELECT * FROM SESSION_ROLES;
ROLE
------------------------------
DBA
SELECT CATALOG ROLE
HS AP FULL DATABASE
GATHER SYSDMIN SELECT ROLE
EXECUTE CATALOG ROLE
HS ADMIN EXECUTE ROLE
DELETE CATALOG ROLE
EXP FULL DATABASE
IMP FULL DATABASE
DATAPUMP EXP FULL DATABASE
DATAPUMP IMTEM STATISTICS
......
已选择 21 行。
```

12.10.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1232-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16854-1-1.html

第 13 章　其他模式对象

模式对象就是存储在用户模式中的数据库对象，在 Oracle 数据库中的模式对象除了表之外还包括索引、临时表、外部表、簇、视图、序列以及同义词等。其中，索引用于提高数据的检索效率；临时表用于存储数据；外部表可以读取操作系统的文件系统中存储的数据；簇可以减少查询数据所需的磁盘读取量；视图用于从一个或多个表（或视图）中导出常用的数据；序列用于自动生成列值；同义词用于为数据库对象定义别名。

本章将详细介绍 Oracle 中的索引、临时表、外部表、簇、视图、序列以及同义词等模式对象的功能以及具体的操作方法。

13.1　使用索引的优势有哪些

13.1.1　问题描述

刚接触 Oracle，对索引不是很了解。今天编写了一条简单的 SQL 语句，查询的字段为 username。当为该字段加上索引之后，发现查询的速度特别快。我想问一下：这是为什么啊？是与索引有关系吗？那么，索引的作用都有哪些？使用它的优势在哪里？

13.1.2　解决方法

索引是建立在表的一列或多个列上的辅助对象，目的是加快访问表中的数据。那么在数据库中为什么要创建索引呢？这是因为，创建索引可以大大提高系统的性能：

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性；

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因；

第三，可以加速表与表之间的连接，特别是在实现数据的参考完整性方面特别有意义；

第四，在使用分组和排序子句进行数据检索时，同样可以减少查询中分组和排序的时间；

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

13.1.3　知识扩展——索引类型

Oracle 中常用的索引类型有：B 树索引、位图索引、反向键索引、基于函数的索引、簇索引和非簇索引 6 种。

1．B 树索引

B 树索引是 Oracle 中默认并且最常用的索引。各叶子节点中包括的数据有索引列的值和数据表中对应行的 ROWID，简单地说，在 B 树索引中，是通过在索引中保存排序后的索引

列值与相对应记录的 ROWID 来实现快速查询的目的。其逻辑结构图如图 13-1 所示。

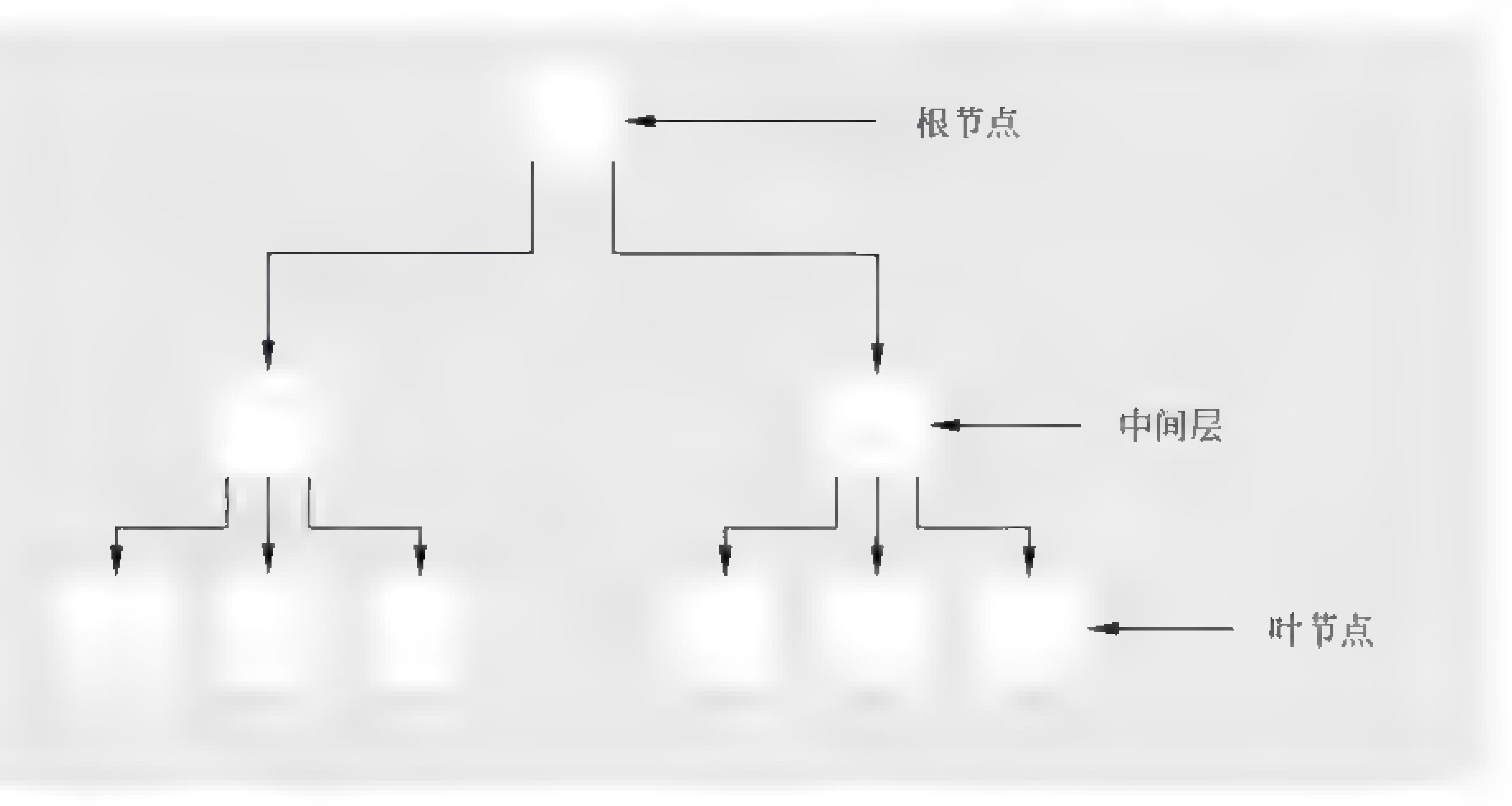


图 13-1　B 树逻辑结构图

B 树是一个多层次、自维护的结构。一个 B 树包括一个顶层，称为根节点（Root Node）、0 到多个中间层（Intermediate）、一个底层（Level 0），底层中包括若干叶子节点（Leaf Node）。在图 13-1 中，每个方框代表一个索引页。索引列的宽度越大，B 树的深度越深，即层次越多，读取记录所要访问的索引页就越多。也就是说，数据查询的性能将随索引列层次数目的增加而降低。



从 B 树索引的逻辑结构图可以看出，B 树索引的组织结构类似于一棵树，其中主要数据都集中在叶子节点上。每个叶子节点中包括：索引列的值和记录行对应的物理地址 ROWID。在 B 树索引中，可以保证无论用户要搜索哪个分支的叶子节点，都需要经过相同的索引层次，即都需要相同的 I/O 次数。

2．位图索引

上面介绍了 B 树索引保存排过序的索引列的值，通过数据行的 ROWID 来实现快速查找。而位图索引既不存储 ROWID 值，也不存储键值，主要用于在比较特殊的列上创建索引。

在 Oracle 中建议，当一个列的所有取值数与表的行数之间的比例小于 1%时，就不适合在该列上创建 B 树索引。

例如，在一个用户注册表中有一列是用户的性别，该列仅有两种取值，分别是：男或女。因此该列上不适合创建 B 树索引，因为 B 树索引主要用于对大量不同的数据进行细分。在该列上使用位图索引的效果如图 13-2 所示。

在图 13-2 中，1 表示“是，该值存在于这一行中”，0 表示“否，该值不存在于这一行中”。虽然 1 和 0 不能作为指向行的指针。但是，由于图表中 1 和 0 的位置与表行的位置是相对应的，如果给定表的起始和终止 ROWID，则可以计算出表中行的物理位置。

在为表中的低基数列创建位图索引时，系统将对表进行一次全面扫描，为遇到的各个取值构建“图表”。例如图 13-2，在图表的顶部列出了两个值：男和女。在创建位图索引进行全表扫描的同时，还将创建位图索引记录，记录中各行的顺序与它在表中的顺序相同。

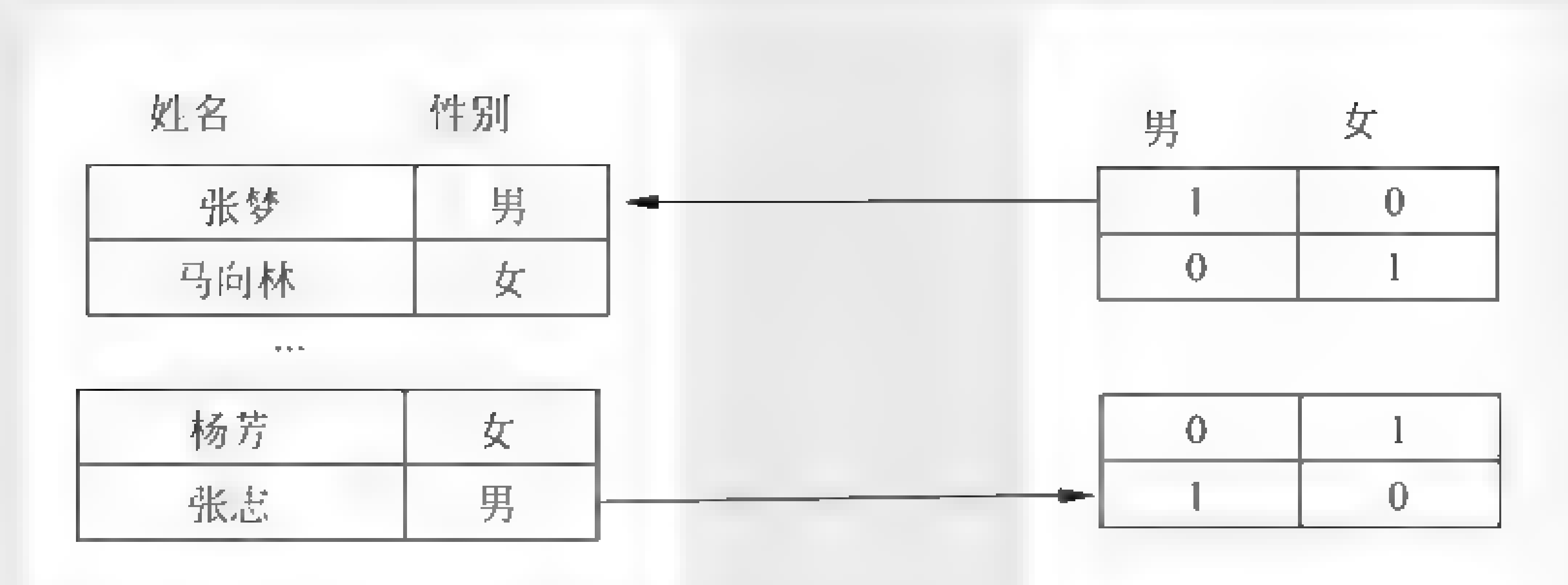


图 13-2　位索引示意图

3．反向键索引

先考虑一个情况：某一字段的值是 1-100 顺序排列，建立 B 树索引后依旧递增，之后该 B 树索引不断在后面增加分支，会形成如图 13-3 的不对称树。

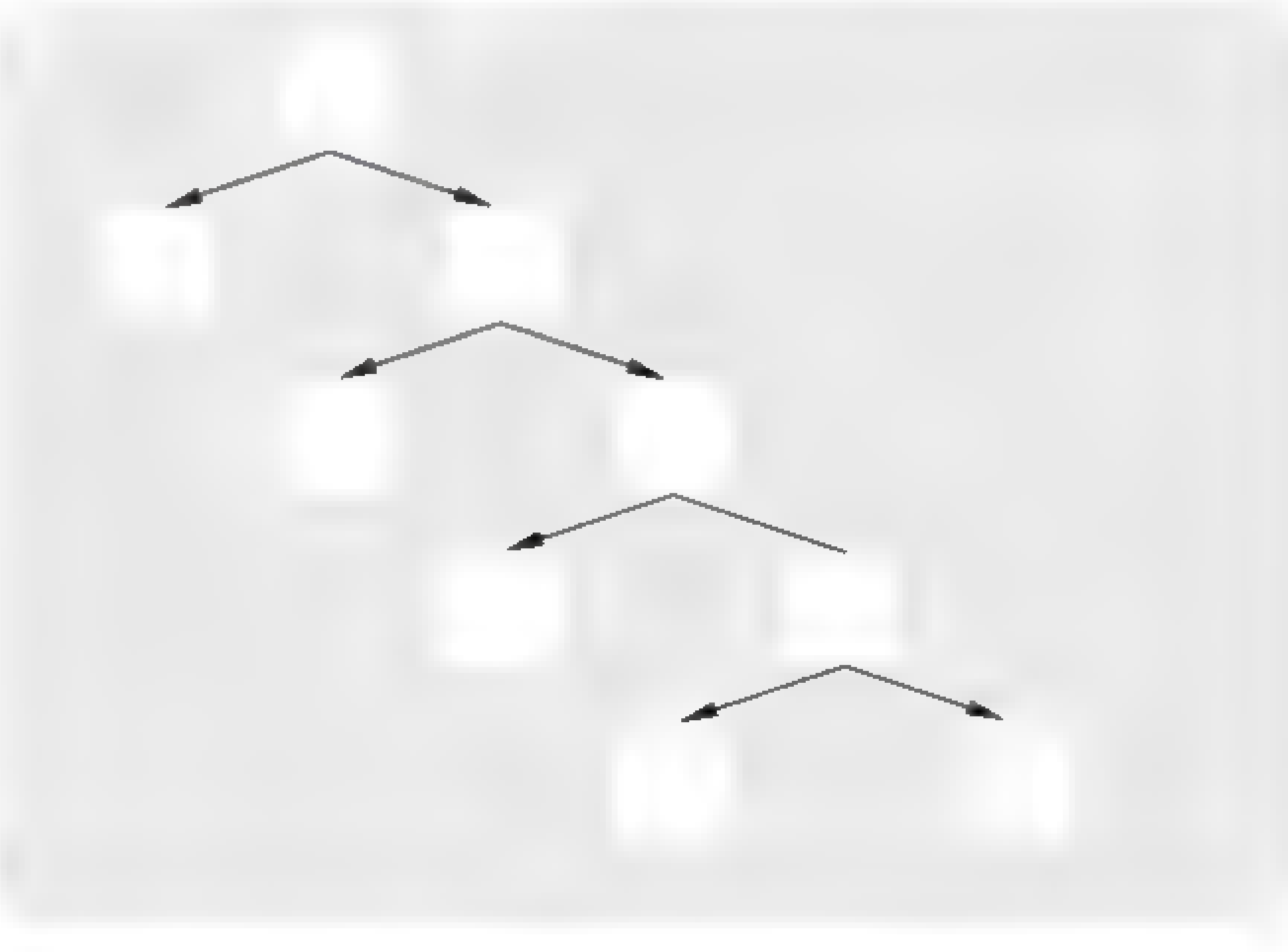


图 13-3　不对称的 B 树

反向键索引是一种特殊的 B 树索引，在存储构造中与 B 树索引完全相同，但是针对数值时，反向键索引会先反向每个键值的字节，然后对反向后的新数据进行索引。例如输入 3005 则转换为 5003，这样当数值依次增加时，其反向键在大小中的分布仍然是比较平均的。

4．基于函数的索引

基于函数的索引只是常规的 B 树索引，只不过它存放的数据是由表中的数据应用函数后得到的，而不是直接存放表中的数据本身。

在 Oracle 中，经常遇到字符大小写或数据类型转换等问题。这时，就可以引用函数对这些数据进行转换。例如，在 SCOTT 模式下的 EMP 表中含有 ENAME 列（在该列上建立了索引），其中一个值为 MAXIANGLIN。如果使用小写的字符串 maxianglin 查找该员工记录，则无法找到。如下：

```
SQL> SELECT empno,ename,sal FROM emp
  2  WHERE ename='maxianglin';
未选定行
```

这时，可以引用函数来解决这个问题。例如，引用 LOWER()函数，将查询时遇到的每个值都转换成小写。查询如下：

```
SQL> SELECT empno,ename,sal
  2  FROM emp
  3  WHERE LOWER(ename)='maxianglin';
EMPNO       ENAME           SAL
----------- --------------- ------
7935        MAXIANGLIN      2975
```

如果即使在 ENAME 列上建立了索引，还是不得不进行全表扫描。在这种情况下，可以使用基于函数的索引，具体的创建方法将在后面的章节中讲解。

5．簇索引

簇索引对表的物理数据页中的数据按列进行排序，然后再重新存储到磁盘上，即簇索引与数据是混为一体的，它的叶节点中存储的是实际的数据。由于簇索引对表中的数据一一进行了排序，因此用簇索引查找数据很快。但由于簇索引将表的所有数据完全重新排列了，它所需要的空间也就特别大，大概相当于表中数据所占空间的 120%。表的数据行只能以一种排序方式存储在磁盘上，所以一个表只能有一个簇索引。

6．非簇索引

非簇索引具有与表的数据完全分离的结构，使用非簇索引不用将物理数据页中的数据按列排序。非簇索引的叶节点中存储了组成非簇索引的关键字和行定位器。行定位器的结构和存储内存取决于数据的存储方式。如果数据是以簇索引方式存储的，则行定位器中存储的是簇索引的索引键；如果数据不是以簇索引方式存储的，这种方式又称为堆存储方式，则行定位器存储的是指向数据行的指针。非簇索引将行定位器按关键字的值用一定的方式排序，这个顺序与表的行在数据页中的排序是不匹配的。

由于非簇索引使用索引页存储，因此它比簇索引需要更多的存储空间，且检索效率较低。但一个表只能建一个簇索引，当用户需要建立多个索引时，就需要使用非簇索引了。从理论上来讲，一个表最多可以建 249 个非簇索引。

13.1.4 触类旁通



Oracle 中更多的索引意味着更高的性能吗？

网络课堂：http://bbs.itzcn.com/thread-16738-1-1.html

在数据库表中创建索引可以加快表的访问速度，并提高数据库的性能。那么是否在一个数据表中创建的索引越多，就意味着该数据库具有越高的性能呢？

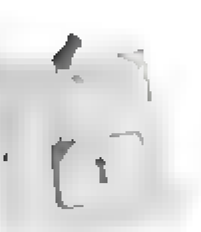
索引是占存储空间的，每一个索引都对应一个索引表，当本表插入数据时会按照一定顺序向索引表中插入一条数据，索引过多会产生以下瓶颈：

（1）占用过多存储空间。

（2）引起插入数据更新、更新数据时速度下降。

因此，索引不是越多越好，过多的话还不如没有索引。总之索引能在一定程度上提高检索速度，特别是表中数据量很大的时候，但是并不是越多越好，切记啊！

13.1.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1242-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-17036-1-1.html

13.2 创建索引出现 ORA-01452 的错误

13.2.1 问题描述

我在实际工作中经常会遇到这样的问题：试图对数据表中的某一列或多列创建唯一索引时，系统提示“ORA-01452：不能创建唯一索引，发现重复记录”。遇到这样的情况，应该如何解决？

13.2.2 解决方法

作为一个 Oracle 数据库开发者或者 DBA，在工作中的确会经常遇到这样的问题，这个问题很容易解决。从错误提示可以看出，因为系统发现表中存在重复的记录，从而系统无法对数据表创建唯一索引，因此我们首先需要找到表中的重复记录并删除其中几条只剩下一条，这样才可以创建唯一索引。

13.2.3 知识扩展——创建索引

在 Oracle 中，索引无论是从逻辑上还是从物理上都不依赖于相应的表，它可以拥有独立的存储空间。创建索引的语法如下：

```
CREATE UNIQUE | BTIMAP
INDEX <schema>.<index name>
ON <schema>.<table name>
(<column name> | <expression> ASC | DESC,
<column name> | <expression> ASC | DESC,…)
TABLESPACE<tablespace name>
STORAGE<storage settings>
LOGGING | NOLOGGING
```

```
COMPUTE STATISTICS
NOCOMPRESS | COMPRESS<nn>
NOSORT | REVERSE
PARTITION | GLOBAL PARTITION<partition_setting>;
```

在上述语法中各关键字或子句的含义如表 13-1 所示。

表 13-1　创建索引时各关键字或子句的含义

关键字或子句	含　义
UNIQUE | BITMAP	在创建索引时，如果指定关键字 UNIQUE，则要求表中的每一行在索引时都包含唯一的值；如果指定 BITMAP 关键字，将创建一个位图索引；如果都省略，则默认创建 B 树索引
ASC	表示该列为升序排列。ASC 为默认排列顺序
DESC	表示该列为降序排列
TABLESPACE	用来在创建索引时，为索引指定存储空间
STORAGE	用户可以使用该子句来进一步设置存储索引的表空间存储参数，以取代表空间的默认存储参数
LOGGING | NOLOGGING	LOGGING 用来指定在创建索引时创建相应的日志记录；NOLOGGING 则用来指定不创建相应的日志记录。默认使用 LOGGING。如果使用 NOLOGGING，则可以更快地完成索引的创建操作，因为在创建索引的过程中不会产生重做日志信息
COMPUTE STATISTICS	用来指定在创建索引的过程中直接生成关于索引的统计信息。这样可以避免以后再对索引进行分析操作
NOCOMPRESS | COMPRESS<nn>	COMPRESS 用来指定在创建索引时对重复的索引值进行压缩，以节省索引的存储空间；NOCOMPRESS 则用来指定不进行任何压缩。默认使用 NOCOMPRESS
NOSORT | REVERSE	NOSORT 用来指定在创建索引时，Oracle 将使用与表中相同的顺序来创建索引，省略再次对索引进行排序的操作；REVERSE 则指定以相反的顺序存储索引值；如果表中行的顺序与索引期望的顺序不一致，则使用 NOSORT 子句将会导致索引创建失败
PARTITION | NOPARTITION	使用该子句，可以在分区表和未分区表上对创建的索引进行分区

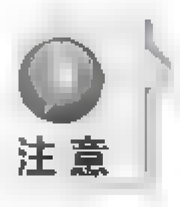
1．创建 B 树索引

例如，为 SCOTT 模式下的 STUDENT 表中的 NAME 列创建一个名称为 stu_index 的索引，如下：

```
SQL> CREATE INDEX stu_index
  2  ON student(name)
  3  TABLESPACE users;

索引已创建。
```

创建索引时，在 ON 关键字后面指定索引基于的表名和列名，使用 TABLESPACE 指定存储索引的表空间。



注意　默认情况下，当用户为表定义一个主键时，系统将自动为该列创建一个 B 树索引。

在创建索引时，应注意下面三点：

- 当一个列已经包含索引时，则无法再在该列上创建索引。
- 默认的索引是不唯一的，但是也可以加上 UNIQUE，表示该索引的字段上没有重复值（定义 UNIQUE 约束时会自动创建）。
- 创建主键时，默认在主键上创建了 B 树索引，因此不能再在主键上创建索引。

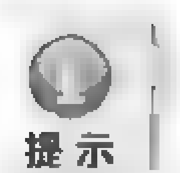
2. 创建位图索引

位图索引适用于在表中基数较小的列上创建。创建位图索引需要使用 BITMAP 关键字，如下：

```
SQL> CREATE BITMAP INDEX stu_bitmap_index
  2  ON student(sex)
  3  TABLESPACE users;

索引已创建。
```

上述语句表示为 STUDENT 表的 SEX 列创建了一个名称为 stu_bitmap_index 的位图索引。



在表上放置单独的位图索引是没有意义的。只有对多个列建立位图索引，系统才可以有效地利用它们提高查询的速度。

位图索引的作用来源于与其他位图索引的结合。当在多个列上进行查询时，Oracle 对这些列上的位图进行布尔 AND 和 OR 运算，最终找到所需要的结果。



位图索引不能是唯一索引，也不能对其进行键压缩。

3. 创建反向键索引

反向键索引适用于在表中严格排序的列上创建。创建反向键索引需要使用 REVERSE 关键字，如下：

```
SQL> CREATE INDEX stu_reverse_index
  2  ON student(id)
  3  REVERSE
  4  TABLESPACE users;

索引已创建。
```

上述语句表示为 STUDENT 表中的 id 列创建了一个名称为 stu_reverse_index 的反向键索引。



键的反转由系统自行完成，用户不需要关心键的反向处理。

4. 创建基于函数的索引

创建基于函数的索引，可以提高在查询条件中使用函数和表达式时查询的执行速度。如

果用户要在自己的模式中创建基于函数的索引，则必须具有 QUERY REWRITE 系统权限；如果用户想要其他模式中创建基于函数的索引，则必须具有 CREATE ANY INDEX 和 GLOBAL QUERY REWRITE 权限。

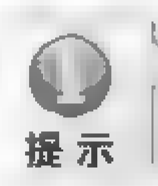
例如，为 SCOTT 模式下的 EMP 表中的 HIREDATE 列创建一个基于函数 TO_CHAR()的函数索引，如下：

```
SQL> CREATE INDEX emp_date_index
  2  ON emp(TO_CHAR(hiredate,'YYYY-MM-DD'))
  3  TABLESPACE users;

索引已创建。
```

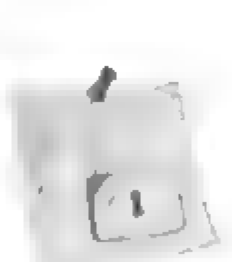
创建该索引后，如果在查询条件中包含有相同的函数，则可以提高查询的执行速度。下面的查询将会使用 emp_date_index 索引：

```
SQL> SELECT empno,ename,hiredate
  2  FROM emp
  3  WHERE TO_CHAR(hiredate,'YYYY-MM-DD')='1987-05-23';
```


提示

创建基于函数的索引时，Oracle 会首先对包含索引列的函数值或表达式值进行求值，然后对求值后的结果进行排序，最后存储到索引中。简单来说，基于函数的索引，就是将查询要用到的表达式作为索引项。

13.2.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1243-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1244-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1245-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1246-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16739-1-1.html

13.3 删除表会删除索引吗

13.3.1 问题描述

想问一下：如果我使用 DROP TABLE 语句删除表，是否会删除该表中的所有索引？因为业务上有几个表每天晚上都会自动重建，即需要先删除该表然后再重新创建。这样优化统计信息是否没有了？需要重新统计，索引是否也需要重建呢？

13.3.2 解决方法

索引是依托在数据表上的，当表不存在时，一切与该表有关的约束、索引……也将被删

除。因此，当使用 DROP TABLE 语句删除数据表时，索引也被删除，需要重新统计数据时，需要重新创建索引。

13.3.3 知识扩展——索引的管理

Oracle 允许我们对于已创建的索引进行管理，例如修改索引的名称、合并索引中的存储碎片、重新创建索引、监视索引的使用情况以及删除不必要的索引等。

1. 修改索引的名称

在 Oracle 中可以将已经创建的索引进行重命名操作，重新命名索引的语法格式如下：

```
ALTER INDEX index_name RENAME TO new_index_name;
```

在上述语法格式中，index_name 代表已定义索引的名称，new_index_name 代表重新命名的索引名称。

例如，将索引名称为 stu_index 重新命名为 stu_b_index，如下：

```
SQL> ALTER INDEX stu_index
  2  RENAME TO stu b index;
索引已更改。
```

2. 合并索引

在实际应用中，表中的数据要不断地进行更新，这会导致在表的索引中产生越来越多的存储碎片，这些碎片会影响索引的使用效率。而合并索引可以清除索引中的存储碎片，其语法格式如下：

```
ALTER INDEX index_name COALESCE [ DEALLOCATE UNUSED ];
```

在上述语法格式中，index_name 表示索引的名称，COALESCE 表示合并索引；DEALLOCATE UNUSED 表示合并索引的同时，释放合并后多余的空间。

合并索引是指将 B 树中叶子节点的存储碎片合并在一起，这种合并不会改变索引的物理组织结构。例如在 B 树中，有多个叶子节点的数据块使用的存储空间为 60%，我们可以使用合并索引的方法来清除索引中的存储碎片。对 stu_b_index 索引进行合并的代码如下：

```
SQL> ALTER INDEX stu_b_index
  2  COALESCE DEALLOCATE UNUSED;

索引已更改。
```

3. 重建索引

除了合并索引可以清除索引中的存储碎片之外，还有一种方式可以清除存储碎片，即重建索引。重建索引在清除存储碎片的同时，还可以改变索引中全部存储参数的设置，以及索引的存储表空间。其语法格式如下：

```
ALTER [ UNIQUE ] INDEX index name
REBUILD;
```

在上述语法格式中，index_name 代表需要重建索引的名称。

例如，重新建立 stu_b_index 索引，如下：

```
SQL> ALTER INDEX stu_b_index
  2  REBUILD;
索引已更改。
```

4．监视索引

监视索引的目的是为了确保索引得到有效的利用，打开索引的监视状态需要使用 ALTER INDEX … MONITORING USAGE 语句，其语法格式如下：

```
ALTER INDEX index_name MONITORING USAGE;
```

在上述语法格式中，index_name 表示索引的名称。

例如，要查看索引 stu_b_index 的使用情况，需要先打开索引的监视状态，然后通过 V$OBJECT_USAGE 动态性能视图查看索引的使用情况，如下：

```
SQL> ALTER INDEX stu_b_index
  2  MONITORING USAGE;
索引已更改。

SQL> SELECT * FROM v$object_usage;
INDEX_NAME    TABLE_NAME    MON  USE    START_MONITORING  END_MONITORING
------------ ----------- ---------- ----------------- ------------
STU_B_INDEX   STUDENT       YES   NO     08/17/2011      09:50:51
```

V$OBJECT_USAGE 动态性能视图的结构如下：

```
SQL> DESC v$object_usage;
 名称                          是否为空?     类型
 ------------------------- ---------- ----------- --------
 INDEX_NAME                    NOT NULL    VARCHAR2(30)
 TABLE_NAME                    NOT NULL    VARCHAR2(30)
 MONITORING                                VARCHAR2(3)
 USED                                      VARCHAR2(3)
 START_MONITORING                          VARCHAR2(19)
 END_MONITORING                            VARCHAR2(19)
```

该视图的字段说明如下：

- **MONITORING**

该字段标识是否激活了使用的监视。

- **USED**

该字段描述在监视过程中索引的使用情况。

- **START_MONITORING 和 END_MONITORING**

这两个字段分别描述监视的开始和终止时间。

如果需要关闭索引的监视状态，需要使用 ALTER INDEX … NOMONITORING USAGE 语句，如下：

```
SQL> ALTER INDEX stu_b_index
  2  NOMONITORING USAGE;
索引已更改。
```



每次使用 MONITORING USAGE 打开索引监视，V$OBJECT USAGE 视图都将针对指定的索引进行重新设置（清除或重新设置以前的使用信息，并记录新的开始时间）；当使用 NOMONITORING USAGE 关闭监视时，则不再执行下一步监视，该监视阶段的结束时间被记录下来。

5．删除索引

当一个索引被删除后，它所占用的盘区会全部返回给它所在的表空间，并且可以被表空间中的其他对象使用。通常在以下情况下需要删除某个索引：

（1）该索引不需要被使用。

（2）该索引很少被使用，索引的使用情况可以通过监视来查看。

（3）该索引中包含较多的存储碎片，需要重建该索引。

用户可以删除在自己模式中的索引，如果要删除在其他模式中的索引，则需要用户必须具有 DROP ANY INDEX 系统权限。其删除索引主要分为如下两种情况：

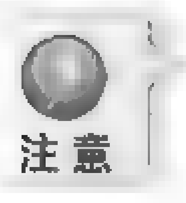
❑ 删除基于约束条件的索引

如果索引是在定义约束条件时由 Oracle 自动建立的（例如定义 UNIQUE 约束时，Oracle 自动创建唯一索引），则必须禁用或删除该约束本身。

❑ 删除使用 CREATE INDEX 语句创建的索引

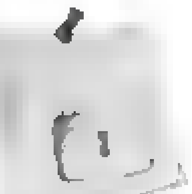
如果索引是使用 CREATE INDEX 语句显示创建的，则需要使用 DROP INDEX 语句删除该索引。例如，删除索引 stu_b_index，如下：

```
SQL> DROP INDEX stu b index;
索引已删除。
```



在删除一个表时，Oracle 会删除所有与该表相关的索引。

13.3.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1247-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1248-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-16740-1-1.html

13.4 为何要使用临时表

13.4.1 问题描述

我创建临时表的 SQL 语句如下：

```
CREATE GLOBAL TEMPORARY TABLE myTable
AS SELECT e.empno,e.ename,e.deptno FROM emp e;
```

这条语句为什么只是创建了一个myTable表的结构，而并没有将emp表中对应列的数据复制过去？如何才能在创建临时表的同时将指定数据表的数据复制到临时表中？

13.4.2 解决方法

在创建临时表时，如果指定了ON COMMIT PRESERVE ROWS，则表示是会话级别临时表，也就是说会话结束后，临时表中的记录会清空。如果不指定，则默认的为ON COMMIT DELETE ROWS，这是事务级别临时表，表示该事务结束后记录便会清空，也就是说当DDL语句（CREATE TABLE...就是一个DDL语句）发出后，Oracle会隐式地提交事务，因此刚刚插入到临时表的数据被自动删除了，这就是你没有查询到数据的原因。

将SQL语句修改为：

```
CREATE GLOBAL TEMPORARY TABLE myTable ON COMMIT PRESERVE ROWS
AS SELECT e.empno,e.ename,e.deptno FROM emp e;
```

这样就可以了，你试试吧！

13.4.3 知识扩展——临时表的类别

由于临时表中存储的数据只在当前事务处理或者会话进行期间有效，因此临时表主要分为两种：事务级别临时表和会话级别临时表，如下：

- 事务级别临时表

创建事务级别临时表，需要使用ON COMMIT DELETE ROWS子句。事务级别临时表的记录会在每次提交事务后被自动删除。

- 会话级别临时表

创建会话级别临时表，需要使用ON COMMIT PRESERVE ROWS子句。会话级别临时表的记录会在用户与服务器断开连接后被自动删除。

13.4.4 知识扩展——创建与使用临时表

Oracle中的临时表是“静态”的，用户不需要在每次使用临时表时重新建立，与普通的数据表一样被数据库保存，并且从创建开始直到被删除期间一直是有效的，被作为模式对象存在数据字典中。通过这种方法，可以避免每当用户应用中需要使用临时表存储数据时必须重新创建临时表。

创建临时表，需要使用CREATE GLOBAL TEMPORARY TABLE语句。临时表主要分为事务级别临时表和会话级别临时表，下面将介绍这两种临时表的创建与使用。

1. 创建与使用事务级别临时表

创建事务级别临时表的语法格式如下：

```
CREATE GLOBAL TEMPORARY TABLE table name ON COMMIT DELETE ROWS;
```

在上述语法格式中，table_name 代表创建的表名称以及字段。

下面的示例演示了如何创建事务级别临时表。其表名为 temp_proc_transcation，该表包含的字段有 id，proname，price 以及 protype：

```
SQL> CREATE GLOBAL TEMPORARY TABLE temp_proc_transcation(
  2   id NUMBER,
  3   proname VARCHAR2(50),
  4   price NUMBER,
  5   protype NUMBER
  6  )
  7  ON COMMIT DELETE ROWS;
表已创建。
```

当创建好临时表之后，就可以使用该临时表了。例如，向临时表 temp_proc_transcation 中插入数据，如下：

```
SQL> INSERT INTO temp_proc_transcation
  2  VALUES(
  3  1,'联想手机',678,2);
已创建 1 行。
```

为了验证插入是否成功，下面我们使用 SELECT 语句查询 temp_proc_transcation 表中的数据，如下：

```
SQL> SELECT * FROM temp_proc_transcation;
ID          PRONAME           PRICE         PROTYPE
---------- ----------------- ------------ --------------
1           联想手机           678            2
```


注意

当使用 COMMIT 提交事务后，再次查询 temp_proc_transcation 表中的数据，会发现数据已经被清空了。这是因为事务级别临时表中的数据在提交事务后被清除了。

2．创建与使用会话级别临时表

创建会话级别临时表的语法格式如下：

```
CREATE GLOBAL TEMPORARY TABLE table_name ON COMMIT PRESERVE ROWS;
```

下面的示例演示了如何创建会话级别临时表。其表名为 temp_proc_session，该表包含的字段与上面创建的事务级别临时表 temp_proc_transcation 中的字段相同，如下：

```
SQL> CREATE GLOBAL TEMPORARY TABLE temp_proc_session(
  2  id NUMBER,
  3  proname VARCHAR2(50),
  4  price NUMBER,
  5  protype NUMBER
  6  )
  7  ON COMMIT PRESERVE ROWS;
表已创建。
```

下面向临时表 temp_proc_session 中插入一条数据，并查询该表中的数据，如下：

```
SQL> INSERT INTO temp_proc_session
  2  VALUES(
  3  1,'摩托罗拉手机',1200,1);
已创建 1 行。

SQL> SELECT * FROM temp_proc_session;
ID          PRONAME              PRICE          PROTYPE
---------- -------------------- -------------- -----------
1           摩托罗拉手机          1200           1
```

与事务级别临时表不同的是，会话级别临时表在提交事务后，temp_proc_session 表中的数据仍然还在，如下：

```
SQL> COMMIT;
提交完成。
SQL> SELECT * FROM temp_proc_session;
ID          PRONAME              PRICE          PROTYPE
---------- ------------------ -------------- -----------
1           摩托罗拉手机          1200              1
```

但是，如果断开与服务器的连接，则该表中的数据会被清除。例如再次连接服务器，并查询 temp_proc_session 表中的数据，如下：

```
SQL> CONNECT SYSTEM/admin;
已连接。
SQL> SELECT * FROM user_session;
未选定行
```



注意 当断开与服务器的连接，然后再次连接服务器，查询会话级别临时表 temp_proc_session 中的数据，会发现数据已经被清空了。这是因为会话级别临时表中的数据在会话断开后会被清除。

13.4.5 触类旁通



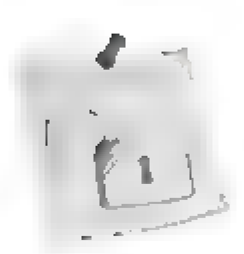
临时表在什么时候创建比较合适？

网络课堂：http://bbs.itzcn.com/thread-16743-1-1.html

在数据管理中，临时表适合在什么情况下使用？在什么时候创建合适呢？可以使用普通表代替临时表吗？

有时候是需要的，比如做报表查询，每个用户都在执行自己的查询，通过临时表就可以把各个用户独立开来。每个用户就感觉只有自己在对临时表进行操作，执行相应的增、删、改、查等操作。如果不用临时表而用普通表，在 Oracle 中每个用户只能靠 SESSION ID 来区分自己和其他人的数据了。

13.4.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-1249-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1250-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-17037-1-1.html

13.5 访问外部表时出错

13.5.1 问题描述

创建目录对象和外部表都成功了，但是在访问外部表时出现如下的错误：

```
ORA-29913: 执行 ODCIEXTTABLEFETCH 调出时出错
ORA-30653: 已达到拒绝限制值
```

这是怎么回事啊？如何解决？

13.5.2 解决方法

这个错误的发生最可能有两个原因：外部文件中的数据类型与你创建的外部表的数据类型不匹配，还有一个原因就是外部文件中分隔之后的字符个数与你创建的外部表的字段个数不匹配。默认情况下，允许出现的错误个数为 0，所以在查询时，提示“已达到拒绝限制值”。

解决的方法有两种：第一种是你仔细检查你编写的外部文件内容分隔之后是否与创建的外部表匹配；另一种是你在创建外部表时，在结尾处添加“REJECT LIMIT UNLIMITED;”选项，表名对错误的个数不限制。

13.5.3 知识扩展——创建外部表

外部表是 Oracle 提供的，它是用于读取操作系统的文件系统中存储数据的一种只读表。使用 Oracle 的外部表可以很容易地将一个格式化的文本文件虚拟成数据库的表，并可以使用 SELECT 语句去访问数据。

创建外部表需要使用 CREATE TABLE … ORGANIZATION EXTERNAL 语句，语法格式如下：

```
CREATE TABLE table_name(…)
ORGANIZATION EXTERNAL (
TYPE ORACLE_LOADER
DEFAULT DIRECTORY directory_name
ACCESS PARAMETERS(
FIELDS TERMINATED BY '分隔符'
```

```
    )
  )
```

在上述语法中，table_name 表示创建的外部表名称，directory_name 表示所创建的目录对象的名称。其参数的含义如下：

- **TYPE**

用来指定访问外部表数据文件时所使用的访问驱动程序，该程序可以将数据从它们最初的格式转换为可以向服务器提供的格式。Oracle 提供的默认访问驱动程序是 ORACLE_LOADER。

- **DEFAULT DIRECTORY**

用来指定所使用的目录对象，该目录对象指向外部数据文件所在目录。

- **LOCATION**

用来指定源数据文件。

- **ACCESS PARAMETERS**

用来设置访问驱动程序进行数据格式转换时的参数。

- **FIELDS TERMINATED BY**

用来指定字段之间的分隔符。

下面介绍如何在 Oracle 中建立外部表去访问一个格式化的文本文件中的数据。

（1）在服务器的 E:\oracle 目录下存在一个名称为 user.csv 的文件，该文件的内容如下：

```
A1230,联想手机,500
B5421,摩托罗拉手机,1200
C5124,诺基亚手机,1800
```



.csv 文件可以使用 Excel 创建，然后在保存时选择保存类型为：CSV（逗号分隔）(*.csv)。上述 user.csv 文件中每一行记录中内容使用英文逗号（,）隔开。

（2）在 Oracle 中，通过使用目录对象作为服务器文件系统上目录的别名。如果用户想要建立指向数据文件位置的目录，则该用户必须具有 CREATE ANY DIRECTORY 权限。下面创建一个名称为 pro_directory 的目录对象，其目录指向服务器的 E:\oracle 目录，也就是 user.csv 文件所在的目录。如下：

```
SQL> CREATE DIRECTORY proc_directory
  2  AS 'E:\oracle\';
目录已创建。
```

（3）创建外部表 product，如下：

```
SQL>  CREATE TABLE product(
  2  prono VARCHAR2(50),
  3  proname VARCHAR2(50),
  4  price NUMBER)
  5  ORGANIZATION EXTERNAL(
  6  TYPE ORACLE_LOADER
  7  DEFAULT DIRECTORY
  8  proc_directory
```

```
  9      ACCESS PARAMETERS(
 10      FIELDS TERMINATED BY ',')
 11      LOCATION ('user.csv')
 12  );

表已创建。
```

上述代码创建了一个外部表 product，该表中的数据来源于外部文件 E:\oracle\user.csv。



注意

在创建表的过程中，如果加上 REJECT LIMIT UNLIMITED 关键字，则可以防止出现错误。

（4）通过 SELECT 语句进行查询，查询外部表 product 中的数据，如下：

```
SQL> SELECT * FROM product;

PRONO              PRONAME                  PRICE
------------------ ------------------------ ------------------
A1230               联想手机                   500
B5421               摩托罗拉手机               1200
C5124               诺基亚手机                 1800
```

13.5.4 触类旁通



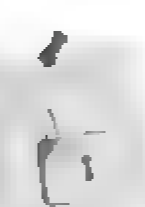
外部表读取外部文件时有什么限制吗？

网络课堂：http://bbs.itzcn.com/thread-16745-1-1.html

通过创建外部表可以读取外部文件上的数据并生成表，那么外部表在读取外部文件时有什么限制吗？

在外部表读取外部文件时，首先要明白，外部表对数据的操作需要数据存放在数据库服务器上或者数据库服务器可以访问的共享路径，需要设置相关参数后才可以进行操作的。外部文件是客户端文本文件，它需要通过一定的规则才可以生成外部表。

13.5.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1251-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16744-1-1.html

13.6 无法删除非空簇

13.6.1 问题描述

簇 sum_cu 中包含有 3 个簇表，我想删除该簇，使用 DROP CLUSTER 语句删除竟然提示

出错，如下：

```
SQL> DROP CLUSTER sum_cu;
DROP CLUSTER sum_cu
*
第 1 行出现错误:
ORA-00951: 簇非空
```

怎么解决啊？删除一个非空的簇要怎么编写 SQL 语句？

13.6.2 解决方法

单单使用 DROP CLUSTER 语句是无法将一个含有簇表的簇删除的，需要使用 DROP CLUSTER … INCLUDING TABLES 语句，SQL 语句如下：

```
DROP CLUSTER sum_cu INCLUDING TABLES;
```

13.6.3 知识扩展——创建簇和簇表

簇由一组共享相同数据块的多个表组成，它将这些表的相关行一起存储到相同数据块中，这样可以减少查询数据所需的磁盘读取量。创建簇后，用户可以在簇中创建表，这些表称为簇表。

如果用户在自己的模式中创建簇和簇表，则必须具有 CREATE CLUSTER 权限和 UNLIMITED TABLESPACE 系统权限；如果在其他模式中创建簇，则还必须具有 CREATE ANY CLUSTER 系统权限。下面将详细介绍如何创建簇和簇表。

1．创建簇

创建簇，需要使用 CREATE CLUSTER 语句。其语法如下：

```
CREATE CLUSTER cluster_name(column data_type[,column data_type] …)
[PCTUSED 40 | integer]
[PCTFREE 10 | integer]
[SIZE integer]
[INITRANS 1 | integer]
[MAXTRANS 255 | integer]
[TABLESPACE tablespace_name]
[STORAGE storage]
```

在上述语法格式中，cluster_name 表示所创建的簇的名称，column 表示对簇中的表进行聚簇存储的字段，data_type 表示该字段的类型。

例如，创建一个名称为 student_mycu 的簇，并指定通过 id 字段来对簇中的表进行聚簇存储，如下：

```
SQL> CREATE CLUSTER student_mycu(id number)
  2  PCTUSED 40
```

```
  3  PCTFREE 10
  4  SIZE 1024
  5  STORAGE(
  6  INITIAL 128K
  7  MINEXTENTS 2
  8  MAXEXTENTS 25)
  9  TABLESPACE users;
簇已创建。
```

在上述代码中，SIZE 子句用来为聚簇字段提供指定的数据块数量。

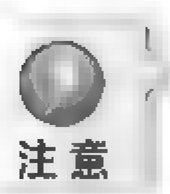
2．创建簇表

创建簇表需要使用 CLUSTER 子句来指定所使用的簇和簇字段。下面通过一个例子来学习如何创建簇表。

在上面创建的 student_mycu 簇中创建一个名称为 student 的簇表，在该簇表中含有 4 个字段，分别为 id、name、age 和 sex。在创建的过程中使用 CLUSTER 子句指定它们所使用的簇为 student_mycu、使用的簇字段为 id。如下：

```
SQL> CREATE TABLE student(
  2  id NUMBER,
  3  name VARCHAR2(50),
  4  age NUMBER,
  5  sex VARCHAR2(2))
  6  CLUSTER student_mycu(id);

表已创建。
```



可以为 student_mycu 簇建立多个簇表，在物理上 Oracle 会将一个簇的多个簇表中相对应的数据存储到相同的数据块中，例如：将学生表和成绩表组成一个簇后，Oracle 会将这两个表中每个学生的基本信息和该学生所有成绩存储到相同的数据块中。

下面我们可以试图向簇表 student 中插入记录，如下：

```
SQL> INSERT INTO student
  2  VALUES(
  3  1,'马向林',22,'女');
INSERT INTO student
            *
第 1 行出现错误:
ORA-02032: 聚簇表无法在簇索引建立之前使用
```

从插入结果来看，现在还无法向簇表中添加记录，为了能够向簇表中添加记录，还需要首先为簇表建立索引。

13.6.4 知识扩展——创建簇索引

簇索引与普通索引一样需要具有独立的存储空间，它与簇表不同，并不存在于簇中。创

建簇索引的语法格式如下：

```
CREATE INDEX index_name
ON CLUSTER clu_name
TABLESPACE <tablespace_name>;
```

在上述语法格式中，index_name 表示创建簇索引的名称，clu_name 表示所创建簇的名称。例如，为簇 student_mycu 建立一个簇索引，然后向使用该簇的簇表中添加记录，如下：

```
SQL> CREATE INDEX mycu_index
  2  ON CLUSTER student_mycu
  3  TABLESPACE users;
索引已创建。
SQL>  INSERT INTO student
  2   VALUES(
  3   1,'马向林',22,'女');

已创建 1 行。
```

13.6.5 知识扩展——管理簇

对于已经创建好的簇，可以对簇进行修改和删除等管理操作。如果用户需要对簇进行管理，必须具有 ALTER ANY CLUSTER 的系统权限。

1. 修改簇

修改一个簇，主要修改簇的三个属性值，分别是：物理存储属性、存储簇键值的所有行所需空间的平均值 SIZE 以及默认的并行度。其中物理存储属性包括 PCTFREE、PCTUSED、INITRANS、MAXTRANS 和 STORAGE。



不能修改存储参数 INITIAL 和 MINEXTENTS。

例如，将簇 student_mycu 中参数 PCTUSED、PCTFREE 和 SIZE 的值分别修改为 50、30 和 1500，如下：

```
SQL> ALTER CLUSTER student_mycu
  2  PCTUSED 50
  3  PCTFREE 30
  4  SIZE 1500;

簇已变更。
```

Oracle 将簇的参数应用于所有属于该簇的簇表，因此对簇的调整也就是对簇表的修改。所以，用户仅可以用 ALTER TABLE 语句进行增加列、修改列等操作。

2. 删除簇

删除簇需要使用 DROP CLUSTER 语句，当一个簇中包含有簇表，单单使用这句话是无

法将簇删除的，下面分别介绍删除一个空簇和非空簇的操作方法。

（1）删除空簇

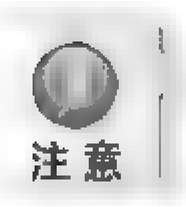
例如簇 myfreecu 不包含任何簇表，删除该簇可以使用 DROP CLUSTER 语句，如下：

```
SQL> DROP CLUSTER myfreecu;
簇已删除。
```

（2）删除含有簇表的簇

对于删除含簇表的簇 student_mycu，需要使用 DROP CLUSTER … INCLUDING TABLES 语句，如下：

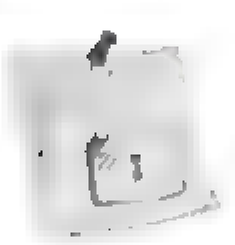
```
SQL> DROP CLUSTER student_mycu
  2  INCLUDING TABLES;
簇已删除。
```



注意　这种删除，实际上就是将簇连同簇中的表一起删除。

另外，如果某个簇含有簇表，并且具有外键约束，则需要使用 DROP CLUSTER … INCLUDING TABLES CASCADE CONSTRAINTS 语句删除该簇；如果需要单独地删除簇中的簇表，可以直接使用 DROP TABLE 语句。

13.6.6　网络课堂



视频教学：http://school.itzcn.com/video-vid-1255-spid-35.html

视频教学：http://school.itzcn.com/video-vid-1256-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16746-1-1.html

13.7　为什么要使用视图

13.7.1　问题描述

在数据库操作中，经常会使用到视图，我想知道的是使用视图都有哪些好处？为什么要使用它？请列举几点视图的用途，让我明白使用视图的好处在哪里。

13.7.2　解决方法

使用视图的好处大致可归结为以下三点：

（1）数据访问控制。注意视图也是一个数据库对象。如果限制用户只能通过视图访问数据，那么就可能限制用户访问指定的数据，而不是数据库中的原始数据。

（2）简单复杂 SQL 的调用。一条 SQL 可能有好多行，通常都是一些报表。直接在 Java 或 C 程序调用并不方便，此时就可以创建一个视图，然后就用一句简单的“SELECT * FROM 视图名”就可以了。

（3）实现相同查询语句的复用。下面讲一个需要统计数据出口的案例。假设大多数的业务都只针对本公司没有离职的员工，每次查询员工时都需要加上条件“WHERE 离职状态=0”，这样很麻烦，也容易因为忘记加上条件而导致出错。所以就可以建立一个视图，这些业务每次查询要处理的员工时，都从视图中查询。当需求改变时，如需要根据出生日期显示员工年龄，也只需要改动视图一处。

13.7.3 知识扩展——管理视图

视图是一个虚拟表，它同真实表一样包含一系列带有名称的列和行数据。但是，视图并不在数据库中存储的数据值，其数据值是来自定义视图的查询语句所引用的表，数据库只在数据字典中存储了视图的定义信息。

视图可以建立在关系表上，也可以在其他视图上，或者同时建立在两者之上。用户可以在视图中进行 INSERT、UDATE 和 DELETE 操作。通过视图修改数据时，实际上就是在修改基本表中的数据。与之相对应，改变基本表中的数据也会反映到由该表组成的视图中。

1．创建视图

在 Oracle 数据库中，创建视图需要使用 CREATE VIEW 语句，其语法格式是：

```
CREATE [OR REPLACE] VIEW <view name>[(ALIAS [,ALIAS]…)]
as<SUBQUERY>;
[WITH CHECK OPTION [CONSTRAIINT constraint name]]
[WITH READ ONLY]
```

在上述语法格式中，ALIAS 用于指定视图列的别名；SUBQUERY 用于指定视图对应的子查询语句；WITH CHECK OPTION 子句用于指定在视图上定义 CHECK 约束；WITH READ ONLY 子句用于定义只读视图。在创建视图时，如果不提供视图列别名，Oracle 会自动使用子查询的列名或列别名；如果视图子查询包含函数或表达式，则必须定义列别名。



如果在当前用户模式中创建视图，那么数据库用户必须具有 CREATE VIEW 系统权限；如果要在其他用户模式中创建视图，则用户必须具有 CREATE ANY VIEW 系统权限。

表视图分为两种创建情况，创建基于一个表的视图和基于多个表的视图。下面详细介绍这两种情况下视图的创建。

（1）基于一个表的视图

假如，创建一个基于 SCOTT.STUDENT 表的视图 stu view，通过该视图可以检索 STUDENT 表中的所有数据。如下：

```
SQL> CONNECT SYSTEM/admin
已连接。
SQL> GRANT CREATE VIEW TO SCOTT;
```

```
授权成功。
SQL> CONNECT SCOTT/tiger;
已连接。
SQL>  CREATE OR REPLACE VIEW stu_view
  2   AS
  3   SELECT * FROM student;
视图已创建。
```

使用 SELECT 语句查询 stu_view 视图中的所有数据，如下：

```
SQL>  SELECT * FROM stu_view;
ID     NAME         AGE          SEX
------ ------------ ------------- --------
1      马向林          22            女
2      殷国鹏          22            男
...
6      白雪            22            女
已选择 6 行。
```



在创建视图时，我们还可以为每个字段指定字段名，例如：CREATE OR REPLACE VIEW stu_view(学号,姓名,年龄,性别)...。实际上，系统在创建视图时，只是将视图的定义存入到数据字典中，并不执行其中的 SELECT 语句。只有当用户对视图进行查询时，系统才按视图的定义从基本表中获取数据。

如果想了解视图的定义信息，可以通过查询数据字典视图 USER_VIEWS 的 TEXT 列，如下：

```
SQL> SELECT text FROM user_views
  2  WHERE view_name='STU_VIEW';
TEXT
--------------------------------------------
select "ID","NAME","AGE","SEX" from student
```

（2）基于多个表的视图

例如，创建一个基于 STUDENT 表与 SCORE 表的视图，如下：

```
SQL> CREATE OR REPLACE VIEW stu_score_view
  2  AS
  3  SELECT stu.id,stu.name,stu.age,stu.sex,sc.result
  4  FROM student stu
  5  INNER JOIN score sc
  6  ON stu.id=sc.stuid;
视图已创建。
```

接着使用 SELECT 语句查询 stu_score_view 视图，如下：

```
SQL> SELECT * FROM stu_score_view;
ID          NAME             AGE              SEX            RESULT
```

```
1           马向林        22          女           89
2           殷国鹏        22          男           90
3           王丽丽        22          女           84
4           马林立        23          女           78
```

2. 更新视图

Oracle 中可以通过更新视图中的数据来修改基本表中的数据。一个视图可以同时包含可更新的字段与不可更新的字段，当视图中的某个字段是由计算获取的，则不予更新，例如下面的视图：

```
SQL> CREATE OR REPLACE VIEW test view(姓名,年龄,性别)
  2   AS
  3   SELECT name,age+3,sex FROM student;
视图已创建。
```

在创建视图 test_view 时，将 STUDENT 表中的字段分别命名为姓名、年龄和性别，即"姓名"对应于 NAME 字段，"性别"对应于 SEX 字段，"年龄"对应于（age+1）字段，也就是说视图中的"年龄"字段是由计算获得的，所以该视图中的"年龄"字段是不可更新的，其他两个字段可更新。

下面我们来更新 test_view 视图中的数据，步骤如下：

（1）查询 test_view 视图数据，并使用 UPDATE 语句对该视图中的字段进行更新操作，如下：

```
SQL> UPDATE test_view SET 姓名='张瑞',性别='男'
  2 WHERE 姓名='马向林';
已更新 1 行。
```

（2）查询 test_view 视图中的数据和 STUDENT 表中的数据是否已经更新成功，如下：

```
SQL> SELECT * FROM test view;

姓名                年龄               性别
--------------- ------------------ --------------------
张瑞                25                 男
殷国鹏              25                 男
王丽丽              25                 女
马林立              26                 女
张小强              28                 男
白雪                25                 女
已选择 6 行。

SQL> SELECT * FROM student;
ID            NAME          AGE               SEX
------------- ------------- ----------------- ----------------
1             张瑞            22                男
```

```
2            殷国鹏        22           男
3            王丽丽        22           女
4            马林立        23           女
5            张小强        25           男
6            白雪          22           女
已选择 6 行。
```

一个视图中的哪些字段是可以更新的，哪些字段是不可以更新的，可以通过查询数据字典视图 USER_UPDATABLE_COLUMNS 视图中来了解。下面来使用该数据字典查询 test_view 视图中的字段哪些是可以更新的，如下：

```
SQL> SELECT column_name,updatable,insertable,deletable
  2  FROM user_updatable_columns
  3  WHERE table_name='TEST_VIEW';

COLUMN_NAME                UPD            INS          DEL
---------------- -------------- -------------- ----------
姓名                          YES            YES          YES
年龄                          NO             NO           NO
性别                          YES            YES          YES
```

视图中的每个字段的 UPDATABLE、INSERTABLE 和 DELETABLE 都包含一个 YES/NO 值，用来表示该字段是否可以执行更新、添加和删除的操作。从查询结果可以看出，视图 test_view 中的姓名和性别字段可以执行更新操作，而年龄字段不可以。



提示

在创建视图时，可以使用 WITH CHECK OPTION 选项，它通常用来将数据从基本表中按一定规范选取出来，当规范定义好之后，用户必须按照这个规范对视图进行操作。例如，在视图中使用 WHERE 子句限定查询条件为“name='马向林'”，则向视图中插入数据时，插入的 NAME 字段的值必须为“马向林”。

3．删除视图

如果要删除其他用户模式中的视图时，要求该用户必须具有 DROP ANY VIEW 系统权限。删除视图，需要使用 DROP VIEW 语句，如下：

```
SQL> DROP VIEW test_view;
视图已删除。
```

执行 DROP VIEW 语句后，视图的定义将被删除，这对视图内所有的数据没有任何影响，它们仍然存储在基本表中。

13.7.4 触类旁通

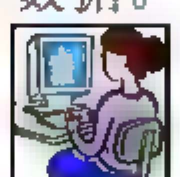


视图中是否保存了所查询的数据。

网络课堂：http://bbs.itzcn.com/thread-16748-1-1.html

使用视图可以方便用户查看相关联表数据，即我们可以使用“SELECT * FROM 视图名称”来查看多个表中的数据，那么视图中是否真的保存了所查询的数据呢？

没有。视图中只是保存了一条 SQL 语句，通过视图所得到的数据，其实也就是查询语句所返回的数据。Oracle 中有一种物化视图，可以将 SQL 所查询的数据保存起来。缺点是会降低数据库增删改的效率，因为在原表数据发生变化时，数据库必须同时更新物化视图中的数据。



通过视图是否可以加快查询速度？

网络课堂：http://bbs.itzcn.com/thread-16749-1-1.html

通过视图我们可以查询到多个表中的数据，是否就意味着视图可以加快查询速度，提高执行效率呢？

通过视图是不能加快查询速度的。当数据量比较大时，可以明显感觉到通过视图查询的效率比直接执行 SQL 要低得多。要提高查询的效率，最简单的方法就是创建合理的索引。

13.7.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1257-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1258-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16747-1-1.html

13.8 Oracle 中序列问题

13.8.1 问题描述

我现在有一个表 table，有 id 和若干个字段，我设置了 id 为自增列：

```
CREATE SEQUENCE res seq
...
START WITH 1
INCREMENT BY 1
...
```

之后我向表中插入了若干条记录，此时 id 是从 1 开始的，正常。但是我将表中的记录删除掉之后又插入若干条记录，发现不是从 1 开始循环了。采取的步骤是这样的：先使用 DROP SEQUENCE 删除序列，然后再次执行上述的创建序列的 SQL 语句，最后刷新服务器。当插入若干条数据之后发现记录也是从 1 开始的，但是不是按 1、2、3、4...这样的顺序排列的，而是 5、4、2、1、3。我不明白这样的情况是怎么回事？是什么引起的？应该如何解决？

13.8.2 解决方法

这和序列的 CACHE 选项有关，默认 CACHE 为 20，也就是每次拿出 20 个序列放到内存

中，当实例崩溃或者内存清洗后则会发生断号的情况。如果你想解决这个问题，可以设置序列的 CACHE 为 1。

13.8.3 知识扩展——管理序列

序列是一数据库对象，使用它可生成唯一的整数，一般使用序列自动地生成主键值。一个序列的值是由特别的 Oracle 程序自动生成，因而序列避免了在运用层实现序列而引起的性能瓶颈。Oracle 序列允许同时生成多个序列号，而每一个序列号是唯一的。当一个序列号生成时，序列是递增的，独立于事务的提交或回滚。允许设置默认序列，不需指定任何子句，该序列为上升序列。

1．创建序列

用户要在自己的模式中创建序列，必须具有 CREATE SEQUENCE 系统权限；如果要在其他模式中创建序列，则必须具有 CREATE ANY SEQUENCE 系统权限。创建序列的语法格式如下：

```
CREATE SEQUENCE sequence name
[START WITH start]
[INCREMENT BY increment]
[MINVALUE minvalue | NOMINVALUE]
[MAXVALUE maxvalue | NOMAXVALUE]
[CACHE cache | NOCACHE]
[CYCLE | NOCYCLE]
[ORDER | NOORDER]
```

语法说明如下：

- **sequence_name** 用来指定待创建的序列名称。
- **START** 用来指定序列的开始位置。在默认情况下，递增序列的起始值为 MINVALUE，递减序列的起始值为 MAXVALUE。
- **INCREMENT** 用来表示序列的增量。该参数值为正数，则生成一个递增序列，为负数则生成一个递减序列。其默认值为 1。
- **MINVALUE** 用来指定序列中的最小值。
- **MAXVALUE** 用来指定序列中的最大值。
- **CACHE | NOCACHE** 用来指定是否产生序列号预分配，并存储在内存中。
- **CYCLE | NOCYCLE** 用来指定当序列达到 MAXVALUE 或 MINVALUE 时，是否可复位并继续下去。如果使用 CYCLE，则如果达到极限，生成的下一个数据将分别是 MINVALUE 或者 MAXVALUE；如果使用 NOCYCLE，则如果达到极限并试图获取下一个值时，将返回一个错误。
- **ORDER | NOORDER** 用来指定是否可以保证生成的序列值是按顺序产生的。如果使用 ORDER，则可以保证；而如果使用 NOORDER，则只能保证序列值的唯一性，而不能保证序列值的顺序。



与视图一样，序列并不占用实际的存储空间，只是在数据字典中保存它的定义信息。

下面通过一个示例演示如何使用序列向已创建好的表中添加数据，步骤如下：

(1)创建一个商品表PRODUCT，该表中含有3个字段：PROID、PRONAME和PROPRICE。如下：

```
SQL>  CREATE TABLE PRODUCT(
  2  PROID NUMBER,
  3  PRONAME VARCHAR2(50),
  4  PROPRICE NUMBER);
表已创建。
```

(2) 创建序列，使其按一定的顺序自动生成PROID字段值，并设置该序列从1开始、每次递增1、没有最大值、不可复位。如下：

```
SQL> CREATE SEQUENCE pro sequence
  2  START WITH 1
  3  INCREMENT BY 1
  4  NOMAXVALUE
  5  NOCYCLE;

序列已创建。
```

(3) 使用序列的伪列NEXTVAL向PRODUCT表中添加新的记录，该伪列用来返回序列生成的下一个值，第一次调用NEXTVAL产生序列的初始值。如下：

```
SQL> INSERT INTO product
  2  VALUES(
  3  pro sequence.nextval,'联想手机',580);
已创建 1 行。

SQL> INSERT INTO product
  2  VALUES(
  3  pro_sequence.nextval,'诺基亚手机',1800);
已创建 1 行。

SQL>  INSERT INTO product
  2   VALUES(
  3   pro sequence.nextval,'摩托罗拉手机',1200);
已创建 1 行。
```

(4) 使用SELECT语句查询PRODUCT表中的数据，如下：

```
SQL> SELECT * FROM product;


PROID           PRONAME                         PROPRICE
```

```
1                   联想手机                        580
2                   诺基亚手机                      1800
3                   摩托罗拉手机                    1200
```



序列除了有一个 NEXTVAL 伪列以外，还有一个 CURRVAL 伪列，该伪列用来返回序列的当前值。例如。序列 pro sequence 的当前值为 3。还有一点需要注意的是：必须在第一次使用 NEXTVAL 之后才能使用 CURRVAL，否则 Oracle 将返回一个错误信息。

2．修改序列

修改序列，需要使用 ALTER SEQUENCE 语句，该语句可以对除了序列的起始值以外的定义序列的任何子句和参数进行修改。如果要修改序列的起始值，则必须先删除该序列，然后重建该序列。

例如，修改序列 pro_sequence 的递增量为 3、序列的最大值为 9999999。如下：

```
SQL> ALTER SEQUENCE pro sequence
  2  INCREMENT BY 3
  3  MAXVALUE 9999999;
序列已更改。
```

接着向 PRODUCT 表中插入一条记录，并查询表中数据，如下：

```
SQL> INSERT INTO product
  2  VALUES(
  3  pro sequence.nextval,'大翼手机',800);
已创建 1 行。

SQL> SELECT * FROM product;
PROID           PRONAME                    PROPRICE
---------- ------------------------ --------------------------
1                联想手机                      580
2                诺基亚手机                    1800
3                摩托罗拉手机                  1200
6                大翼手机                      800
```

3．删除序列

使用 DROP SEQUENCE 语句可以删除序列，如下：

```
SQL> DROP SEQUENCE pro sequence;
序列已删除。
```

删除序列时，Oracle 只是将它的定义从数据字典中删除。

13.8.4 触类旁通



Oracle 中序列与字段的问题。

网络课堂：http://bbs.itzcn.com/thread-16751-1-1.html

在 Oracle 中新建了一个序列，我想插入一条数据就必须把 SQL 语句写成：INSERT INTO aa(id,name) VALUES (序列名.nextval,'aa')，我想问一下，有没有方法可以把插入的那个序列与 id 列进行绑定，这样就可以使用下面的语句进行插入操作了：

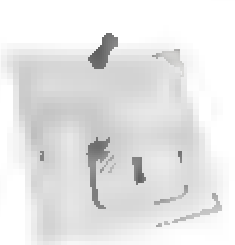
```
INSERT INTO aa(name) VALUES('aa');
```

将 id 与序列号进行绑定需要使用触发器来实现，如下：

```
CREATE OR REPLACE TRIGGER bef T text
BEFORE INSERT ON 表名
FOR EACH ROW
BEGIN
SELECT 序列名.nextval INTO :new.绑定字段名 FROM dual;
END;
```

运行这段代码就可以了，并且绑定的字段可以不是主键。

13.8.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1259-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1260-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16750-1-1.html

13.9 对其他模式的数据库进行操作时遇到的麻烦

13.9.1 问题描述

在 Oracle 中对用户的管理是使用权限的方式来管理的，也就是说，如果我们想使用数据库，我们就必须得有权限，但是如果是别人将权限授予了我们，我们也是能对数据库进行操作的，但是我们必须要在已授权的表的名称前键入该表所有者的名称，所以比较麻烦，遇到这种情况，我们该怎么办呢？

13.9.2 解决方法

这种问题对于一个数据库管理员来说是经常会遇到的。在 Oracle 系统中，可以使用同义词来解决此类问题。同义词其实就是给其他模式的数据库对象创建了一个别名，只需要通过同义词名称就可以访问其他模式的数据库对象了。

13.9.3 知识扩展——管理 Oracle 同义词

Oracle 数据库中提供了同义词管理的功能。同义词是数据库方案对象的一个别名，经常

用于简化对象访问和提高对象访问的安全性。在使用同义词时，Oracle 数据库将它翻译成对应方案对象的名字。与视图一样，同义词并不占用实际存储空间，只有在数据字典中保存了同义词的定义。在 Oracle 数据库中的大部分数据库对象，如表、视图、同义词、序列、存储过程、包等等，数据库管理员都可以根据实际情况为他们定义同义词。

Oracle 同义词有两种类型，分别是公用 Oracle 同义词与私有 Oracle 同义词：

- **公有同义词**　由一个特殊的用户组 PUBLIC 所拥有。顾名思义，数据库中所有的用户都可以使用公用同义词。公用同义词往往用来标示一些比较普通的数据库对象，这些对象往往大家都需要引用。
- **私有同义词**　它与公用同义词所对应，由创建它的用户所有。当然，这个同义词的创建者，可以通过授权控制其他用户是否有权使用属于自己的私有同义词。

1．创建同义词

创建同义词的语法如下：

```
CREATE [PUBLIC] SYNONYM synonym_name
FOR schema_object
```

其中，PUBLIC 关键字用来指定创建的同义词是否为公有同义词；synonym_name 为创建的同义词名称；schema_object 为同义词所针对的对象。

例如为 SCOTT 模式下的 DEPT 表对象创建一个名称为 dept_public 的同义词，并使用 PUBLIC 关键字指定其为共有同义词，这样可以在其他用户模式中通过同义词的名称来访问 SCOTT.DEPT 表中的数据，如下：

```
SQL> CONNECT SYSTEM/admin;
已连接。
SQL>  CREATE PUBLIC SYNONYM dept_public
  2   FOR SCOTT.dept;

同义词已创建。
```

下面使用 SYS 用户登录，并通过同义词 dept_public 来访问 SCOTT.DEPT 表中的数据，如下：

```
SQL> CONNECT SYS/admin AS SYSDBA;
已连接。
SQL> SELECT * FROM dept_public;

DEPTNO         DNAME          LOC
-------------  -------------  -------------

10             ACCOUNTING     NEW YORK
20             RESEARCH       DALLAS
30             SALES          CHICAGO
40             OPERATIONS     BOSTON
```



创建同义词时，所对应的模式对象不必存在。

2．删除同义词

使用 DROP SYNONYM 语句可以删除同义词。如果是删除公有同义词，则还需要指定 PUBLIC 关键字。如下：

```
SQL> DROP PUBLIC SYNONYM dept public;
同义词已删除。
```

13.9.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1261-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16752-1-1.html

第 14 章　大　对　象

众所周知，数据库的作用是用来存储数据的，通常情况下存储的数据都是一些比较简单的数据信息。在之前要想将图片或者视频等信息存储到数据库中，通常就是以地址字符串的形式进行存储，如果文件丢失那么就等于该数据信息也丢失了。但是对于 Oracle 大型数据库来说存储这些数据有一套属于自己的方法，在 Oracle 数据库中提供了一种数据类型 LOB（Large Objects，大对象），该类型可以将图片或者音频文件以二进制的方式存入数据库中，那么就不怕数据文件的丢失了。

接下来将会介绍 LOB 类型的应用以及操作和常用的方法。

14.1　TO_BLOB()函数有什么用

14.1.1　问题描述

今天在对一批 SQL 语句进行操作时，出现运行异常。当开始检查代码是否有误时，发现在一条插入的 SQL 语句中有 TO_BLOB()的一个函数，不知道该函数是否有用处？是不是因为该函数导致的运行异常？大家帮我解答下，具体代码如下所示。

```
INSERT INTO adjunct VALUES(1,'内容',TO_BLOB('123456'))
```

14.1.2　解决方法

当数据库表中有 BLOB 数据类型时，那么必须通过使用该函数将普通数据类型转为 BLOG 数据类型才可以插入到数据库内。

你可以检查下创建的表中对应的列是否为 BLOG 数据类型。如果是 BLOG 数据类型那么必须使用该函数进行转换，而普通数据不可以插入到该列中。

14.1.3　知识扩展——Oracle 大对象（LOB）简介

LOB（Large Object）是 Oracle 数据库用于存储大对象的数据类型。该数据类型主要存储二进制数据、字符数据、引用外部文件的指针的数据类型，例如医学记录（如 X-射线）、视频、图像等内容。

表 14-1 是 LOB 对象常用的 4 种数据类型。

表 14-1　LOG 常用类型

类型	说明
CLOB	字符 LOB 类型，用于存储字符数据
NCLOB	国家语言字符集 LOB 类型，用于存储多字节字符数据（一般用于非英文字符）
BLOB	二进制 LOB 类型，用于存储二进制数据
BFILE	二进制 FILE 类型，用于存储文件指针。有时候，数据文件本身可以存储在数据库之外，而在数据库中只存储对该文件的引用

在上表中，BFILE 类型是 LOB 对象中存储内容最大的类型，它可以容纳 4GB 的数据。CLOB、NCLOB 和 BLOB 三种类型可以存储（4GB-1）*DB_BLOCK_SIZE 数据，其中 DB_BLOCK_SIZE 为系统参数值最大为 32KB，因此计算下来这三种类型最大可以存储 128TB（1TB=1204GB）的数据信息。

14.1.4　知识扩展——BLOB 数据类型

BLOB（Binary Large Objects）数据类型为二进制对象。在其他常用的数据库中，例如 Access 数据库、Sybase 数据库和 SQL Server 数据库中并不都叫做 BLOB 类型，它们都有各自的叫法，但是从广义上讲，它们都属于 BLOB 类型。

BLOB 类型可以分为三种存储形式：声像数据、二进制数据和大文本数据。最常用的就是将图片或者音频对象存储到该类型列中。在将普通数据存储到该类型列时需要使用函数将数据进行转换后才可以放入该列。

在创建表时，为某列添加 BLOB 数据类型的操作非常简单。例如以下代码是在创建 test 表时为该表 images 列设置为 BLOB 数据类型。

```
SQL> CREATE TABLE test(
  2  id NUMBER NOT NULL,
  3  images BLOB NOT NULL
  4  );
表已创建。
```

14.1.5　触类旁通



如何修改 BLOB 类型数据列数据？

网络课堂：http://bbs.itzcn.com/thread-16856-1-1.html

在 test 数据库中，image 列的数据类型为 BLOB 数据类，当修改该列数据值时出现异常信息，如何解决代码如下所示。

```
SQL> update test set images=123123;
update test set images=123123
                *
第 1 行出现错误:
ORA-00932: 数据类型不一致: 应为 BLOB, 但却获得 NUMBER
```

在上述代码中执行了修改语句，将 images 列的值修改为 123123，可是执行时提示数据类型不一致，如何解决这样的问题。

修改数据语法没有问题，主要是因为数据类型不匹配。在创建的 test 表中，images 列的数据类型为 BLOB 类型，而修改的数据类型为 NUMBER 类型，因此出现数据类型不一致的错误，正确的修改方法如下所示。

```
SQL> update test set images=to blob('123132');

已更新 1 行。
```

14.1.6 网络课堂

视频教学：http://school.itzcn.com/video-vid-1271-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1273-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-16855-1-1.html

14.2 Oracle CLOB 字段插入问题

14.2.1 问题描述

大家都知道 CLOB 列是用来存储大对象数据的，那么在使用 INSERT 语句对该类型列插入数据时出现异常，代码如下所示。

```
SQL> INSERT INTO test clob VALUES(1,itzcn);
INSERT INTO test clob VALUES(1,itzcn)
                              *
第 1 行出现错误:
ORA-00984: 列在此处不允许
```

在为 CLOB 类型类插入数据时，提示这样的错误，是否和插入的数据类型有关系？还是其他原因造成的？

14.2.2 解决方法

在向 CLOB 类型列添加数据时，同样要求数据类型相互对应。通常使用 TO CLOB()函数将普通数据转换为 CLOB 类型数据，然后进行插入，正确的代码如下所示。

```
SQL> INSERT INTO test_clob VALUES(1,TO_CLOB('itzcn'));

已创建 1 行。
```

将普通的数据用 TO_CLOB()函数转换过之后就可以插入到 CLOB 类型数据列中了。

14.2.3 知识扩展——创建包含 CLOB 数据列的表

字符 LOB 类型属于大对象中的数据类型之一，用于存储字符数据。该类型的创建也非常简单，以下代码就是在创建表时为表中某列添加 CLOB 数据类型。

```
SQL> CREATE TABLE test_clob(
  2  id NUMBER NOT NULL,
  3  content CLOB NOT NULL
  4  );

表已创建。
```

以上代码是在创建 test_clob 表时，为该表 content 列添加 CLOB 数据类型。

除此之外对 CLOB 类型数据还可以进行增、删、改、查等数据操作，通过这些操作能够更加方便的管理 CLOB 数据。

1. 插入 CLOB 数据

在向 CLOB 类型数据列插入数据时，需要使用 INSERT 语句来完成，例如以下插入代码。

```
SQL> INSERT INTO test_clob(id,content)
  2  VALUES(1,to_clob('我是 CLOB 数据'));
已创建 1 行。
```

在上述代码向 test_clob 表中插入 CLOB 类型数据时，需要使用 TO_CLOB()函数对其数据进行类型转换方可插入该列。

2. 查询 CLOB 数据

查询 CLOB 数据也非常简单，和查询普通字符串类型数据相同需要使用 SELECT 语句来完成，例如以下代码。

```
SQL> SELECT * FROM test_clob;
ID          CONTENT
------      ---------------
1           我是 CLOB 数据
```

3. 更改 CLOB 数据

在对 CLOB 类型数据类进行修改和插入时都需要使用到 TO_CLOB()函数来对数据进行类型转换，然后再使用 UPDATE 进行修改操作。

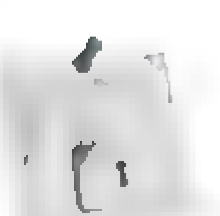
```
SQL> UPDATE test_clob SET
  2  content=to_clob('CLOB 被修改了') WHERE id=1;
已更新 1 行。
```

4. 删除 CLOB 数据

对 CLOB 数据进行删除时，只需要使用 DELETE 语句即可完成。代码如下。

```
SQL> DELETE FROM test clob WHERE id=1;
已删除 1 行。
```

14.2.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1272-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16857-1-1.html

14.3 如何将图片保存在 Oracle 数据库中

14.3.1 问题描述

在实际开发过程中，经常会将一些文件或者图片存储在数据库中，这样方便管理员对图片的管理和操作。那么在 Oracle 数据库中，如何将一张或者多张图片存储在数据库中？

14.3.2 解决方法

在 Oracle 数据库中有 BFILE 类型，该类型主要用于存储指向文件的指针。在存储文件指针前需要为数据库创建一个目录对象，将文件存储在该目录下，代码如下。

```
SQL> CREATE DIRECTORY IMAGES AS 'D:\images';

目录已创建。
```

然后，创建一个表，并且为该表的某列设置为 BFILE 类型，详细代码如下：

```
SQL> CREATE TABLE test bfile(
  2  id NUMBER NOT NULL,
  3  images BFILE NOT NULL
  4  );

表已创建。
```

上述代码中，创建了名为 test bfile 的表，并且为该表中的 images 列添加了 BFILE 数据类型。

接下来开始向表中插入图片信息。因为 images 列的数据类型为 BFILE，该数据类型中存储的值为一个外部文件的指针，因此在添加数据时使用 BFILENAME()函数来生成指针，代码如下所示。

```
SQL> INSERT INTO test bfile VALUES(1,BFILENAME('IMAGES','top.jpg'));

已创建 1 行。
```

在使用 BFILENAME()函数时，需要为该函数指定两个参数，第一个为目录对象的名称；第二个参数表示文件名和后缀，并且该文件应该存放在 D:\images 目录下。

这样就将一个图片信息存储到了 test_bfile 表中，如果需要查看该类型的数据可以运行以下代码。

```
SQL> SELECT * FROM test_bfile;

ID          IMAGES
----------  ----------------------------------------
1           bfilename('IMAGES', 'top.jpg')
```

14.3.3 知识扩展——BFILE 数据类型

BFILE 数据类型主要用于存储文件的指针，在数据库中并不存在该文件，不过通过服务器系统可以访问到该文件那么就需要在数据库服务器上创建目录对象。

而目录对象表示文件在文件系统中的存储目录，通常通过使用 CREATE DIRECTORY 语句创建文件目录对象，创建语法如下所示。

```
CREATE DIRECTORY bfile_name AS file
```

语法中 bfile_name 表示目录对象名称；file 表示创建的存储目录位置。

在填充数据时，需要使用 BFILENAME()函数来获取外部文件的指针，语法如下所示。

```
INSERT INTO table_name VALUES([BFILENAME('bfile_name','file_name')])
```

语法中，为一个 INSERT 语句，其中 table_name 表示插入数据的表名称；bfile_name 表示目录对象名称；file_name 表示文件的名称包含后缀。

目录对象创建完成后就需要创建一个拥有 BFILE 数据类型列，具体创建代码如下：

```
SQL> CREATE TABLE test_bfile(
  2  id NUMBER PRIMARY KEY,
  3  content BFILE NOT NULL);
表已创建。
```

在上述代码中，就成功地在 test_bfile 表中创建了一个 BFILE 类型的数据列 content。

表创建完成了，对表数据的操作也存在增加、修改、查询和删除等操作，下面就是对 BFILE 类型数据进行管理操作。

1．增加 BFILE 数据

在对 BFILE 类型列插入数据时就需要使用目录对象，在前面已经讲解了目录对象创建的方法，在这里就需要创建一个目录对象然后再向该类型列中插入数据，代码如下。

```
SQL> CREATE DIRECTORY file_bfile AS 'd:\mydb\bfile';
目录已创建。
SQL> INSERT INTO test_bfile(id,content)
  2  VALUES(1,bfilename('file_bfile','index.html'));
已创建 1 行。
```

在上述代码中，首先创建了名称为 file_bfile 的目录对象，该目录对象实际路径为 d:\mydb\bfile。目录创建完成，开始向 test_bfile 表中插入数据，在插入 BFILE 类型数据时需要使用 BFILENAME()函数来进行数据的转换，该函数有两个参数：第一个参数为目录对象名称，在代码中为 file_bfile；第二个参数表示存储的文件名称。

2．查询 BFILE 数据

对 BFILE 类型数据进行查询操作和查询普通类型数据相同，使用 SELECT 语句即可完成。代码如下。

```
SQL> SELECT * FROM test_bfile;
ID            CONTENT
-- -----        -- - ---- -----
1             bfilename('file_bfile','index.html')
```

3．修改 BFILE 数据

执行修改 BFILE 类型数据时也需要使用 BFILENAME()函数对修改的新数据进行类型转换，然后方可更新到该列中。

```
SQL> UPDATE test_bfile SET
  2  content=bfilename('file_bfile','1.txt')
  3  WHERE id=1;
已更新 1 行。
```

4．删除 BFILE 数据

对 BFILE 类型数据删除只需要 DELETE 语句即可完成，不需要使用任何转换，代码如下。

```
SQL> DELETE FROM test_bfile WHERE id=1;
已删除 1 行。
```

14.3.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1274-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16858-1-1.html

14.4 如何比较两个 LOB 类型数据

14.4.1 问题描述

在 Oracle 数据中，它自身拥有很多的包。其中 DBMS_LOB 包中包含了很多对 LOB 数据操作的方法。由于项目的需求要对数据库中的两对 LOB 数据进行比较，那么在 DBMS_LOB

包中是否有对 LOB 数据比较的方法？如果有那么如何来实现两个 LOB 对象数据的比较。

14.4.2 解决办法

的确，在 DBMS_LOB 包中拥有很多操作 LOB 数据的方法。例如读取 LOB 中的数据方法、复制 LOB 数据到另外一个 LOB 中、将数据从一个 LOB 复制到文件中等。其中 COMPARE() 方法就是用来比较两个 LOB 中存储的数据。

在前面的实例中已经成功地创建了 test_clob 表，该表中 content 列为 CLOB 类型，以下代码首先向该表中插入两条测试数据，代码如下。

```
SQL> INSERT INTO test_clob VALUES(1,TO_CLOB('www.itzcn.com'));
已创建 1 行。

SQL> INSERT INTO test_clob VALUES(2,TO_CLOB('www.itzcn.net'));
已创建 1 行。
```

测试数据插入成功之后开始使用 COMPARE() 方法比新插入的两条数据，先创建 test_compare 存储过程，在过程中对两条数据进行比较代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE test_compare
  2  AS
  3  clob_lob1 CLOB;
  4  clob_lob2 CLOB;
  5  BEGIN
  6  SELECT content INTO clob_lob1 FROM test_clob WHERE id=1 FOR UPDATE;
  7  SELECT content INTO clob_lob2 FROM test_clob WHERE ID=2 FOR UPDATE;
  8  DBMS_OUTPUT.PUT_LINE(DBMS_LOB.COMPARE(clob_lob1,clob_lob2,4,1,1));
  9  END;
 10  /

过程已创建。
```

在上述代码中成功地创建了 test_compare 存储过程。在该存储过程中声明了 clob_lob1 和 clob_lob2 两个变量，用于存储读取出来的两条 CLOB 数据。然后对两条数据进行比较，最后输出比较结果。接下来则是执行过程。

```
SQL> SET SERVEROUTPUT ON;
SQL> EXEC test_compare;
0

PL/SQL 过程已成功完成。
```

运行该存储过程后，输出值为 0，说明比较的两个数据在有效偏移量内相配。在存储过程中 COMPARE()方法设置的偏移量从 1 开始比较 4 为字符。因此输出的结果为 0。

14.4.3 知识扩展——DBMS_LOB 包中常用方法

在 Oracle 数据库中，DBMS_LOB 包包含了很多操作 LOB 数据的方法和函数。通过这些方法可以有效地对 LOB 数据进行读取、修改、比较以及其他的操作。下面介绍几个常用的方法。

1．APPEND()方法

APPEND()方法是将参数中一个值拼接补充到另外一个参数值尾端。以下语法则是该方法的语法结构，其中该方法有两种语法结构：

```
DBMS_LOB.APPEND(
    dest_lob IN OUT NOCOPY BLOB,
    src_lob IN BLOB
);

DBMS_LOB.APPEND(
    dest_lob IN OUT NOCOPY CLOB/NCLOB CHARACTER SET ANY_CS,
    src_lob IN CLOB/NCLOB CHARACTER SET dest_lob%CHARSET
);
```

两个语法功能相同，只不过操作的数据有些区别。语法中 dest_lob 表示将要拼接的参数；src_lob 表示被拼接的值。第二段语句中 CHARACTER SET ANY_CS 表示 dest_lob 中的数据可以为任何字符集；



在使用 APPEND()方法时，方法中两个参数不可以为空，否则将会抛出异常。

例如在下面实例代码中，对 test_append 表中 content 列数据进行拼接，具体实现步骤如下。

（1）在上节中已经讲解了创建 CLOB 类型表的方法，并借助 test_clob 表进行测试，首先需要向该表中插入测试数据，代码如下。

```
SQL> INSERT INTO test_clob(id,content)
  2  VALUES (1,to_clob('第一次测试'));
已创建 1 行。
SQL> SELECT * FROM test_clob;
ID             CLOB_TABLE
-------------- ------------------
1              第一次测试
```

（2）创建过程，对 test_clob 表中 content 列数据使用 APPEND()方法进行数据拼接，实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u_append
  2  AS
```

```
3 str content CLOB;
4 BEGIN
5 SELECT content INTO str content FROM test clob WHERE id-1
6 FOR UPDATE;
7 DBMS LOB.APPEND(str conetnt,to clob('第二次测试'));
8 END;
9 /
过程已创建。
```

上述代码中，创建了 u_append 过程，在该过程中使用 APPEND()方法对 test_clob 表中的 CLOB 类型数据进行拼接操作。

（3）过程创建完成后，调用该过程，然后再查询该表中数据的变化代码如下。

```
SEL> EXEC u append;
PL/SQL 过程已成功挖成。
SQL> SELECT * FROM test_clob;
ID          CONTENT
---------   -----------------
1           第一次测试第二次测试
```

2. CLOSE()方法

该方法理解起来非常简单，从词义可以得知该方法用于关闭已经打开的 LOB，其语法如下所示。

```
DBMS LOB.CLOSE(
    lob IN OUT NOCOPY BLOB | CLOB | NCLOB | BFILE
);
```

在语法中只有一个参数 lob 表示将要被关闭的 LOB 对象。

3. COMPARE 方法

COMPARE 方法用于对两个 LOB 数据进行比较，比较的方式是从指定的偏移量开始，对指定数量的字符或者字节进行比较。详细语法如下。

```
DBMS LOB.COMPARE(
    lob1 IN BLOB | CLOB | NCLOB | BFILE,
    lob2 IN BLOB | CLOB | NCLOB | BFILE,
    amount IN INTEGER,
    offset1 IN INTEGER,
    offset2 IN INTEGER
) RETURN INTEGER;
```

该语法中 lob1 和 lob2 表示将要进行比较的 LOB 数据；amount 表示将要比较的字符数量；offset1 和 offset2 参数表示比较的两个 LOB 中的字符偏移量，也就是从第几位字符开始比较。

该函数运行后将会返回一个 INTEGER 类型值，如果为 1 表示两个 LOB 不相同，如果返回值为 0 则表示比较的两个 LOB 数据相同。

例如下面实例代码中，实现对两个 CLOB 类型数据的比较，实现步骤如下。

（1）创建存储过程，在该过程中实现对两个 CLOB 数据比较的操作，然后将其比较结果输出。

```
SQL> CREATE OR REPLACE PROCEDURE u_complare
  2  AS
  3  dest_clob CLOB :=to_clob('第一次测试');
  4  src_clob CLOB := to_clob('第二次测试');
  5  messages INTEGER;
  6  BEGIN
  7  messages :=DBMS_LOG.COMPARE(dest_clob,src_clob,5,1,1);
  8  DBMS_OUTPUT.PUT_LINE('结果为：' || messages);
  9  END;
 10  /
过程已创建。
```

在上述代码中，分别定义了两个 CLOB 类型数据，然后又定义了用于输出结果的 messages 变量，在过程体内使用 COMPARE()方法对两个 CLOB 数据进行比较。

（2）运行该过程，将比较结果打印在控制台中，代码如下。

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC u_complare;
结果为：-1
PL/SQL 过程已成功完成。
```

4．COPY()方法

该方法用于对 LOB 数据进行复制操作，主要是将一个 LOB 数据复制到另外一个 LOB 中，不过要求从 COPY()方法参数偏移量开始复制执行字符数。语法如下所示。

```
DBMS_LOB.COPY(
    dest_lob IN OUT NOCOPY BLOB | CLOB | NCLOB,
    src_lob IN BLOB | CLOB | NCLOB,
    amount IN INTEGER,
    dest_offset IN INTEGER,
    src_offset IN INTEGER
);
```

其中，dest_lob 表示将要被复制到的位置；src_lob 表示被复制的 LOB 数据值；amount 表示复制的字符数；dest_offset 和 src_offset 表示复制的两个 LOB 偏移量。

下面实例代码中创建一个名为 u_copy 过程使用 COPY()方法对数据进行复制操作，实现步骤如下。

（1）创建 u_copy 过程，代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u_copy
  2  AS
  3  dest_clob OUT NOCOPY CLOB;
  4  src_clob CLOB := to_clob('第一次测试');
```

```
  5  BEGIN
  6  DBMS LOB.COPY(dest clob,src clob,5,1,1);
  7  DBMS OUTPUT.PUT LINE('复制后的结果：' || dest clob);
  8  END;
  9  /
```

在上述代码中，将 src clob 变量中的数据复制到 dest clob 变量中，然后输出到控制台中。

（2）运行该过程，执行代码如下。

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC u copy;
复制后的结果：第一次测试
PL/SQL 过程已成功完成。
```

5. CREATETEMPORARY()方法

CREATETEMPORARY()方法在用户默认的临时表空间中创建 LOB，该方法语法如下所示。

```
DBMS LOB.CREATETEMPORARY(
    lob IN OUT NOCOPY BLOB | CLOB | NCLOB,
    cache IN TRUE | FALSE,
    duration IN PLS INTEGER
);
```

其中，lob 表示将要创建的 LOB；cache 表示创建的 LOB 是否读入缓冲区缓存，该参数可选值为 TRUE 和 FALSE；参数 duration 表示是否删除临时 LOB，该参数可选值为 DBMS_LOB.SESSION、DBMS_LOB.TRANSCATION 或 DBMS_LOB.CALL。

在下面实例代码中，创建一个 u_createtemporary 过程，在该过程中实现创建临时表空间，实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u createtemporary
  2  AS
  3  BEGIN
  4  DBMS LOB.CREATETEMPORARY(lobs,FALSE,DBMS LOB.SESSION);
  5  END;
  6  /
```

6. ERASE()方法

该方法用于删除一个 LOB 中的数据。删除的方式是从指定的偏移量开始，删除指定数量的字符或字节。详细语法如下：

```
DBMS LOB.ERASE(
    lob IN OUT NOCOPY BLOB | CLOB | NCLOB,
    amount IN OUT NOCOPY INTEGER,
    offset IN INTEGER
);
```

该语法中，lob 表示被删除的 LOB 数据；amount 表示删除字符的个数；offset 表示开始删除的索引，也就是偏移量。

在下面实例中，将删除 CLOB 中的部分数据。并且输出删除后的结果，实现步骤如下。

（1）创建 u_erase 过程，在该过程中使用 REASE()方法对数据进行删除操作，代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u_erase
  2  AS
  3  str_clob CLOB := to_clob('第一次测试');
  4  BEGIN
  5  DBMS_LOB.ERASE(str_clob,3,1);
  6  DBMS_OUTPUT.PUT_LINE('删除后结果：' || str_clob);
  7  END;
  8  /
```

（2）调用该过程，将删除后的结果进行输出。

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC u_erase;
删除后的结果：测试
PL/SQL 过程已成功完成。
```

7．FILEGETNAME()方法

该方法理解起来也非常简单，就是用来获取 BFILE 的目录和文件名称，其语法如下所示。

```
DBMS_LOB.FILEGETNAME(
    bfile IN BFILE,
    directory OUT VARCHAR2,
    filename OUT VARCHAR2
);
```

语法中，bfile 表示在数据库中指向文件的指针；directory 表示存储文件的目录；filename 表示文件的名称。



使用该方法时，参数 directory 和 filename 不可以为空值。

8．ISTEMPORARY()方法

ISTEMPORARY()方法作用是检查 LOB 是否是一个临时的 LOB，该方法语法如下所示。

```
DBMS_LOB.ISTEMPORARY(
    lob IN BLOB | CLOB | NCLOB
) RETURN INTEGER;
```

如果检查的结果是临时的 LOB，那么将会返回 1；如果检查 LOB 不是一个临时的 LOB 就返回 0。

9. TRIM()方法

TRIM()用于将读取的LOB数据按照指定长度截取，该方法使用的语法如下所示。

```
DBMS LOB.TRIM(
    lob IN OUT NOCOPY BLOB | CLOB | NCLOB,
    new length IN INTEGER
);
```

其中语法中lob表示将要被截取的LOB数据对象；new length表示将要被截取的长度，该长度以字符为单位。

10. FILECLOSEALL()方法

FILECLOSEALL()方法用于关闭所有BFILE对象。该方法的使用语法如下：

```
DBMS_LOB.FILECLOSEALL;
```

例如在下面实例代码中，创建一存储过程，在执行该过程时关闭所有的BFILE对象，代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u filecloseall
  2  AS
  3  BEGIN
  4  DBMS.LOB.FILECLOSEALL;
  5  END;
  6  /
过程已创建。
```

11. FILEEXISTS()方法

FILEEXISTS()方法用于检查外部文件是否存在，如果检查该文件存在那么将会返回1，如果检查文件不存在那么返回0。该方法使用语法如下。

```
DBMS_LOB.FILEEXISTS(bfile IN BFILE)RETURN INTEGER;
```

上述语法中bfile参数表示外部文件BFILE。

在下面实例中，通过检查test表中的BFILE指针所指的外部文件是否存在，然后将检查的结果输出到控制台中，同样实现该功能需要创建一存储过程，代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u fileexists
  2  bf BFILE;
  3  ms INTEGER;
  4  BEGIN
  5  SELECT content INTO bf FROM test WHERE id=2 FOR UPDATE;
  6  ms := DBMS_LOG.FILEEXISTS(bf);
  7  DBMS OUTPUT.PUT LINE(ms);
  8  END;
  9  /
过程已创建。
```

在上述代码中，将查询出 test 表中的 BFIILE 类型数据赋予给 bf 变量，然后使用 DBMS LOG.FILEEXISTS()方法检查该数据指针指定的文件是否存在，然后将返回结果输出，下面代码为运行后输出结果。

```
SQL> EXEC u fileexists;
0
PL/SQL 过程已成功完成。
```

最后该过程运行结果输出 0 表示该 BFILE 数据指针文件不存在。

12．FREETEMPORARY()方法

FREETEMPORARY()方法用于释放用户默认临时表空间中的临时 LOB。该方法的使用语法有两种，分别针对 BLOB 和 CLOB，如下：

```
DBMS LOB.FREETEMPORARY(
    lob IN OUT NOCOPY BLOB
);

DBMS LOB.FREETEMPORARY(
    lob IN OUT NOCOPY CLOB/NCLOB CHARACTER SET ANY_CS
);
```

该语法中有 1 个参数表示需要释放的临时 LOB。

13．INSTR()方法

该方法用于返回 LOB 数据中，从指定偏移量开始匹配指定次数字符的位置。该方法只有一个返回值，用于返回匹配字符的位置，如果没有返回值则将会返回 0。下面三种语法是针对 BLOB、CLOB 和 BFILE 数据类型指定的语法。

```
DBMS LOB.INSTR(
    lob IN BLOB,
    pattern IN RAW,
    offset IN INTEGER,
    n IN INTEGER
) RETURN INTEGER;

DBMS LOB.INSTR(
    lob IN CLOB/NCLOB CHARACTER SET ANY CS,
    pattern IN VARCHAR2 CHARACTER SET lob%CHARSET,
    offset IN INTEGER,
    n IN INTEGER
) RETURN INTEGER;

DBMS LOB.INSTR(
    bfile IN BFILE,
    pattern IN RAW,
    offset IN INTEGER,
```

```
    n IN INTEGER
) RETURN INTEGER;
```

在上述语法代码中，lob 和 bfile 表示被读取的 LOB 数据；pattern 表示字符串匹配模式；offset 表示偏移量值；n 表示匹配次数。

例如下面实例代码中，创建名称为 u_instr 过程，在该过程中查询 test_clob 表中数据并且在执行匹配次数后显示匹配字符的位置，实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u_inster
  2  cb CLOB;
  3  ms INTEGER;
  4  BEGIN
  5  SELECT content INTO cb FROM test_clob WHERE id=1 FOR UPDATE;
  6  ms := DBMS_LOB.INSTR(cb,'z',1,2);
  7  DBMS_OUTPUT.PUT_LINE('检查结果：' || ms);
  8  END;
  9  /
过程已创建。
```

在上述代码中，创建了 cb 用于检查的 CLOB 变量和返回结果值的 ms 变量。首先将查询 test_clob 表中的数据赋值给 cb，然后使用 DBMS_LOB.INSTR()方法进行检查在 cb 变量中第二次出现“z”字母的位置，然后将其结果输出。下面代码为运行结果。

```
SQL> EXEC u_inster;
检查结果：10
PL/SQL 过程已成功完成。
```

14. ISOPEN()方法

ISOPEN()方法用于检查指定的 LOB 是否已经打开。如果检查中 LOB 已经被打开，那么将返回 1，如果没有被打开将会返回 0。该方法有 3 种语法分别针对 BLOB、CLOB 和 BFILE 类型数据进行操作，语法如下。

```
DBMS_LOB.ISOPEN(
    lob IN BLOB
) RETURN INTEGER;

DBMS_LOB.ISOPEN(
    lob IN CLOB/NCLOB CHARACTER SET ANY_CS
) RETURN INTEGER;

DBMS_LOB.ISOPEN(
    bfile IN BFILE
) RETURN INTEGER;
```

在语法中，lob 表示被检查的 LOB；bfile 表示指向要检查的外部文件的 BFILE。

下面例子中在存储过程 u_isopen 过程中实现查询 test_clob 表中的 CLOB 是否被打开，然

后将结果输出。实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u isopen
  2  cb CLOB;
  3  ms INTEGER;
  4  BEGIN
  5  SELECT content INTO cb FROM test clob WHERE id=1 FOR UPDATE;
  6  ms := DBMS LOB.ISOPEN(cb);
  7  DBMS OUTPUT.PUT LINE('结果: ' || ms);
  8  END;
  9  /
过程已创建。
```

过程创建完成后，使用 EXEC 命令执行 u_isopen 过程，其运行结果如下。

```
SQL> EXEC u  isopen;
检查结果: 0
PL/SQL 过程已成功完成。
```

15. OPEN()方法

该方法用于打开一个 LOB，分别可以打开 BLOCB、CLOB 和 BFILE 类型数据，下面语法就是对着三种数据操作的不同代码。

```
DBMS LOB.OPEN(
    lob IN OUT NOCOPY BLOB,
    open mode IN BINARY INTEGER
);

DBMS LOB.OPEN(
    lob IN OUT NOCOPY CLOB/NCLOB CHARACTER SET ANY CS,
    open mode IN BINARY INTEGER
);

DBMS LOB.OPEN(
    bfile IN OUT NOCOPY BFILE,
    open mode IN BINARY INTEGER
);
```

上述语法中，lob 表示将要被打开的 LOB；bfile 表示将要被打开的外部文件 BFILE；open mode 表示打开模式，该参数默认值为 DBMS LOB.FILE READONLY，即只可读取 LOB。可选值还有 DBMS_LOB.FILE READWRITE，即可以读写 LOB。

下面实例代码就是打开一个 CLOB 类型数据，具体实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE u open
  2  cb CLOB;
  3  BEGIN
  4  SELECT content INTO cb FROM test_clob WHERE id=1 FOR UPDATE;
```

```
  5 DBMS LOB.OPEN(cb,DBMS LOB.FILE READWRITE);
  6 END;
  7 /
过程已创建。
```

在上述代码中，创建过程 u open，将 test clob 表中的 CLOB 数据查询出来然后使用 DBMS LOB.OPEN()方法进行打开操作。

16. READ()方法

READ()方法从词面意义上理解是读取的意思，在程序中它同样充当着这样的角色，只不过该方法是将 LOB 类型数据读取到缓冲区中，该方法使用语法如下。

```
DBMS LOB.READ(
    lob IN CLOB/NCLOB CHARACTER SET ANY CS,
    amount IN OUT NOCOPY BINARY INTEGER,
    offset IN INTEGER,
    buffer OUT VARCHAR2 CHARACTER SET lob%CHARSET
);
```

上述语法代码是针对 CLOB 或 NCLOB 类型数据进行读取的语法，同样也可以对 BLOB 和 BFILE 类型数据进行读取。该语法中 lob 参数表示将要写入的 LOB 数据；amount 表示写入最大字节数；offset 表示写入时的偏移量；buffer 表示存储写入 LOB 中的数据的变量。

17. SUBSTR()方法

SUBSTR()方法用于对 LOB 数据进行截取操作，该方法使用时可以对 BLOB、CLOB 和 BFILE 类型数据进行截取，其语法如下。

```
DBMS LOB.SUBSTR(
    lob IN BLOB,
    amount IN INTEGER,
    offset IN INTEGER
) RETURN RAW;

DBMS LOB.SUBSTR(
    lob IN CLOB/NCLOB CHARACTER SET ANY CS,
    amount IN INTEGER,
    offset IN INTEGER
) RETURN VARCHAR2 CHARACTER SET lob%CHARSET;

DBMS LOB.SUBSTR(
    bfile IN BFILE,
    amount IN INTEGER,
    offset IN INTEGER
) RETURN RAW;
```

上述代码中，lob 表示将要被截取的 LOB 数据；bfile 表示被截取的外部文件的 BFILE；amount 表示截取最大字符数；offset 表示截取的偏移量。该方法有一个返回值，如果截取的

是 BLOB 或 BFILE，则返回 RAW 类型数据；如果截取的是 CLOB 或 NCLOB，则返回 VARCHAR2 类型数据；

在下面实例代码中将会创建 u_substr 存储过程，在过程中截取 CLOB 数据，将截取后的结果输出到控制台，实现代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE u_substr
  2  cb CLOB;
  3  ms VARCHAR2;
  4  BEGIN
  5  SELECT content INTO cb FROM test_clob WHERE id=1 FOR UPDATE;
  6  ms := DBMS_LOB.SUBSTR(cb,6,1);
  7  DBMS_OUTPUT.PUT_LINE('截取结果：' || ms);
  8  END;
  9  /
```

上述代码中，在 u_substr 过程中将 test_clob 中的 CLOB 类型数据进行截取，从第 1 个字符开始截取到第 6 个字符结束，然后将结果输出。下面代码为输出结果。

```
SQL> EXEC u_substr;
截取结果：第一次
PL/SQL 过程已成功完成。
```

18．WRITE()方法

WRITE()方法用于将缓冲区中的数据写入到 LOB，该方法只对 BLOB 和 CLOB 类型数据有效，下面是该方法的使用语法。

```
DBMS_LOB.WRITE(
    lob IN OUT NOCOPY BLOB,
    amount IN BINARY_INTEGER,
    offset IN INTEGER,
    buffer IN RAW
);

DBMS_LOB.WRITE(
    lob IN OUT NOCOPY CLOB/NCLOB CHARACTER SET ANY_CS,
    amount IN BINARY_INTEGER,
    offset IN INTEGER,
    buffer IN VARCHAR2 CHARACTER SET lob%CHARSET
);
```

在上述代码中，参数 lob 表示写入数据的 LOB；amount 表示写入的最大字符数；offset 表示写入时的偏移量；buffer 用于存储写入 LOB 中的数据的变量。

14.4.4 触类旁通



使用 ERASE()方法删除 LOB 数据问题？

网络课堂：http://bbs.itzcn.com/thread-662-1-1.html

在 Oracle 数据库的 DBMS LOB 包中 ERASE()方法是用于删除 LOB 的数据，可是不知为何在执行删除时出现以下异常信息，是否是因为 LOB 数据不可使用该方法删除？

```
SQL> CREATE OR REPLACE PROCEDURE test erase
  2  AS
  3  clob content CLOB;
  4  BEGIN
  5  SELECT content INTO clob content FROM test clob WHERE id=1 FOR UPDATE;
  6  DBMS LOB.ERASE(clob content,4);
  7  END test_erase;
  8  /
```

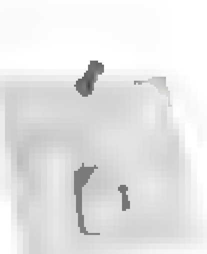
以上代码是创建一个名为 test_erase 的存储过程，该过程用于删除 test_clob 表中 ID 为 1 的 CLOB 数据的前 4 个字符，当运行该过程时出现异常，什么原因？

在使用 ERASE()方法删除 LOB 数据时，该方法的三个参数一个不可以缺少，而在你编写的代码中缺少一个偏移量。正确的代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE test erase
  2  AS
  3  clob content CLOB;
  4  BEGIN
  5  SELECT content INTO clob content FROM test clob WHERE id=1 FOR UPDATE;
  6  DBMS LOB.ERASE(clob content,4,1);
  7  END test erase;
  8  /
```

这样在使用该方法删除 LOB 数据时就可以从读取的数据第一位删除到第四位了。

14.4.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1275-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1276-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1277-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1278-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16859-1-1.html

14.5 LONG 和 LONG RAW 类型是否还可以使用

14.5.1 问题描述

今天在做项目时，发现数据库中出现 LONG 和 LONG RAW 数据类型。可当初看过一本书上说这两种数据类型已经被 Oracle 放弃，所以现在很少人使用过这两种数据类型。不知道

现在是否还可以使用这两种类型，今天借此机会希望大家回答下，小弟在这里谢谢了！

14.5.2 解决方法

在 Oracle 数据库中的确已经放弃了 LONG 和 LONG RAW 数据类型，不过现在还可以使用这两种数据类型。之所以可以继续使用被放弃的数据类型，是因为能够向后兼容，放弃的东西不代表不可以使用。

下面代码则是创建两个具有 LONG 和 LONG RAW 数据类型的表，详细代码如下所示。

```
SQL> CREATE TABLE test_long(
  2  id NUMBER NOT NULL,
  3  content LONG NOT NULL
  4  );
表已创建。

SQL> CREATE TABLE test_long_raw(
  2  id NUMBER NOT NULL,
  3  content LONG RAW NOT NULL
  4  );
表已创建。
```

在上述代码中，分别创建了 test_long 和 test_long_raw 两个表，并且为两表中 content 列数据类型分别设置为 LONG 和 LONG RAW 类型。表创建成功之后，为了确保该数据类型还可以继续使用，可以为新创建的 test_long 和 test_long_raw 两个表中插入测试数据进行测试，详细代码如下所示。

```
SQL> INSERT INTO test_long VALUES(1,'123456');
已创建 1 行。

SQL> INSERT INTO test_long_raw VALUES(1,'10101011010');
已创建 1 行。
```

代码中分别为 test_long 和 test_long_raw 两个表中插入了两条不同的数据，并且提示已经成功地插入说明该数据类型还可以使用。

14.5.3 知识扩展——LONG 和 LONG RAW 数据类型

在前面学习过 LOB 大块数据类型，可是在学习时可能会遇到下面几种和 LOB 非常相似的数据类型 LONG、LONG RAW 和 RAW。

在 Oracle 数据库中，除了前面讲解的数据类型之外，这三种数据类型也可以存储大文件对象。其中：

- ❑ LONG 数据类型可以存储 2GB 以下的字符数据。
- ❑ LONG RAW 数据类型可以存储 2GB 以下的二进制数据。

❑ RAW 数据类型可以存储 4KB 的二进制数据。

LONG 和 LONG RAW 数据类型只支持向后兼容现有的应用程序。一个 LONG 或 LONG RAW 列的最大数据容量为 2GB。

14.5.4　触类旁通



LONG 或 LONG RAW 类型数据转换为 LOG。

网络课堂：http://bbs.itzcn.com/thread-16861-1-1.html

在操作数据时，可能有时会要求将 LONG 和 LONG RAW 数据类型的值转换为新的 LOG 类型，因为 LOG 类型与 LONG 和 LONG RAW 数据类型几乎相似，只不过 LOG 类型是它们的新类型，功能和兼容方面比 LONG 和 LONG RAW 类型好，那么如何实现数据之间的转换？

在 Oracle 系统中，可以将 LONG 或 LONG RAW 类型转换为 CLOB 和 BLOB 类型，转换的方式有两种：一种使用 TO_LOB()函数，另外一种则直接使用 LATER TABLE 语句来完成。

1．使用 TO_LOB()函数

下面代码是通过使用 TO_LOB()将 LONG 和 LONG RAW 类型数据转换并且存储到其他表中，详细代码如下所示。

```
SQL> INSERT INTO test_clob SELECT id,TO_LOB(content) FROM test_long;
已创建 1 行。

SQL> INSERT INTO test_blob SELECT id,TO_LOB(content) FROM test_long_raw;
已创建 1 行。
```

通过使用 TO_LOB()函数已经成功的将 test_long 和 test_long_raw 表中的数据插入到了对应类型表中。

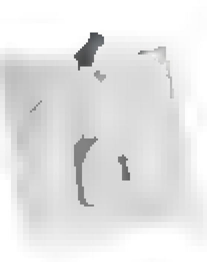
2．使用 ALTER TABLE 语句

使用 ALTER TABLE 语句对 LONG 和 LONG RAW 类型数据进行转换，方法非常简单，只不过他的转换方式是把表中的数据类型修改为了 BLOB 或者 LONG RAW。具体实现代码如下所示。

```
SQL> ALTER TABLE test_long MODIFY(content CLOB);
表已更改。

SQL> ALTER TABLE test_long_raw MODIFY(content BLOB);
表已更改。
```

14.5.5　网络课堂



视频教学：http://school.itzcn.com/video-vid-3977-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16860-1-1.html

14.6 LOB类型数据是否可以加密

14.6.1 问题描述

数据的加密一直以来都是比较关注的问题，因为对数据加密后，可以防止为授权用户对数据的查看和修改。例如通常会对用户密码或者其他用户私密信息进行加密。那么是否可以对 Oracle 数据库中的大对象数据类型 LOB 类型数据进行加密，如何实现加密？

14.6.2 解决办法

在 Oracle 数据库系统中，可以为大对象 LOB 数据类型进行加密操作，不过在加密之前需要创建一个“电子钱包（wallet）”存储安全细节。具体创建步骤如下所示。

（1）在 Oracle 数据安装目录 orcl 下创建 wallet 目录，例如 F:\app\administrator\admin\orcl\wallet 目录。

（2）然后运行命令为电子钱包设置密码，具体命令如下所示。

```
SQL> CONNECT system/tiger
已连接。

SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "123456";
系统已更改。
```

在该命令执行完成之后将会在创建的 wallet 目录下自动生成 ewallet.p12 文件。

（3）在创建表时，为 CLOB 类型数据列设置加密功能，设置加密时需要使用 ENCRYPT 和 SECUREFILE 关键字，详细代码如下所示。

```
SQL> CREATE TABLE test_wallet(
  2  id NUMBER NOT NULL,
  3  content CLOB ENCRYPT USING 'AES192'
  4  )LOB(content) STORE AS SECUREFILE
  5  (CACHE);

表已创建。
```

上述代码中创建了 test_wallet 表，并且在该表中设置了 CLOB 类型类，对该列的数据进行加密。

（4）表创建成功之后开始对表进行数据的添加，添加时因为 content 列类型是 CLOB 类型所以还需要通过使用 TO_CLOB()函数对数据进行转换，代码如下所示。

```
SQL> INSERT INTO test_wallet VALUES(1,'itzcn');

已创建 1 行。
```

此时，插入到 test wallet 表中的 content 列数据就会在后台进行加密操作，当用户使用 SELECT 语句在 SQL*Plus 中查看该表 content 列数据时，系统将会自动将其进行解密返回到控制台中。

14.6.3 知识扩展——创建电子钱包

在对数据进行加密之前，数据库管理员必须先创建电子钱包，其中电子钱包中有加密数据用的密钥信息。

在创建电子钱包之前需要在$ORACLE_BASE\admin\$ORACLE_SID 目录下创建 walle 目录，$ORACLE_BASE 表示 Oracle 数据库安装目录；$ORACLE_SID 表示创建电子钱包的数据的系统标识符。创建语法如下所示。

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BU password
```

语法中，password 表示电子钱包的管理密钥密码。当成功创建电子钱包之后，将会在$ORACLE_BASE\admin\$ORACLE_SID 目录下创建 ewallet.p12 的文件。

同时数据库管理员可以对电子钱包进行开启和关闭操作，默认新创建的电子钱包状态为开启状态，关闭后用户将无法访问和操作加密的数据列。开关电子钱包语法如下所示。

```
ALTER SYSTEM SET WALLET OPEN | CLOSE IDENTIFIED BY password
```

14.6.4 知识扩展——加密 LOB 数据

Oracle 数据库中，可以对 LOB 对象数据进行加密操作，其中包含 BLOB、CLOB 和 NCLOB 数据类型。但是 BFILE 类型不可以进行加密操作，因为该类型存储的为文件指针地址，而这些文件是在数据库之外的。

对大对象类型数据还有不同的加密方式，下面就是不同算法来对数据进行不同的加密方法。

- **3DES168** 密码长度为 168 比特的三重数据加密标准算法。
- **AES128** 密码长度为 128 比特的高级加密标准算法。
- **AES192** 密码长度为 192 比特的高级加密标准算法。
- **AES256** 密码长度为 256 比特的高级加密标准算法。

以下语法则是创建表时，为 LOG 数据列进行加密的语法，在加密时需要使用 ENCRYPT 和 SECUREFILE 关键字，详细代码如下所示。

```
CREATE TABLE table name(
column name CLOB | BLOB | NCLOB ENCRYPE [USING algorithm]
)LOG(column name) STORE AS SECUREFILE
(CACHE READS | CACHE | NOCACHE)
```

其中，table name 表示创建的表名称；column name 表示列名称；CLOB|BLOB|NCLOB 表示可以对着三种数据类型进行加密；algorithm 表示加密的算法；CACHE

READS|CACHE|NOCACHE 表示缓冲器高缓选项。

14.6.5 触类旁通



使用 SQL*Plus 查询加密数据出错!

网络课堂：http://bbs.itzcn.com/thread-16863-1-1.html

当我利用 SQL *Plus 命令查询数据时，本已经加密的数据出现了异常。代码如下所示，大家帮我研究下。

```
SQL> SELECT id,content FROM test wallet;
ERROR:
ORA-28365: Wallet 未打开
```

可是当我查询没有加密的数据时还可以操作，代码如下所示。

```
SQL> SELECT id FROM test wallet;


ID
----------
1
```

从提示信息上不难看出是因为你的电子钱包没有处于打开状态。当电子钱包处于关闭状态时，还可以检索和修改为加密的数据内容，而加密的数据必须当电子钱包处于打开状态时才可以操作。

解决方法首先需要使用 DBA 登录，使用命令打开电子钱包，具体代码如下所示。

```
SQL> CONNECT system/tiger
已连接。

SQL> ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY "123456";
系统已更改。
```

运行以上代码就可以重新打开电子钱包，用户就可以对加密的数据进行搜索和修改等操作，如果需要关闭运行以下代码。

```
SQL> ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY "123456";

系统已更改。
```

14.6.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-3978-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16861-1-1.html

第 15 章　Oracle SQL 语句优化

在 Oracle 中执行 SQL 语句时，为了提高运行效率、减轻 Oracle 的负担，我们需要在编写 SQL 语句时遵循一些使用规则，这些使用规则在数据量少的情况下并不能体现出比较明显的改善，但随着数据量的增加，这些小小的细节将会提高数据库的性能，节省 SQL 语句的执行时间。SQL 语句的优化就是将性能较低的 SQL 语句转换成达到同样目的的性能优异的 SQL 语句。本章将就 Oracle SQL 语句的优化进行讲解，包括一些基本的 SQL 语句编写技巧、索引的巧妙使用和连接查询应该注意的事项等。

15.1　如何提高大数据量查询效率

15.1.1　问题描述

一个有百万条记录的表，一条简单的 SQL 语句：

```
SELECT * FROM tablename ORDER BY time;
```

执行上述语句需要花费 42 秒的时间。我这里其实就是查询时间最近的一百条记录，有什么方法可以优化 SQL 语句并获取前 100 条记录的方法吗？

15.1.2　解决方法

如果这个表数据量非常多，而这些语句又必须频繁执行，那么需要将“*”替换为表 tablename 中的字段，并采用隐含字段 rownum，例如：

```
SELECT column1,column2,column3 ... FROM tablename
WHERE rownum<101
ORDER BY time DESC;
```

15.1.3　知识扩展——避免使用“*”替代所有列

在使用 SELECT 语句查询一个表的所有列信息时，可以使用动态 SQL 列引用“*”，用来表示表中所有的列。使用“*”替代所有的列，可以降低编写 SQL 语句的难度，减少 SQL 语句的复杂性，但是却降低了 SQL 语句执行的效率。

下面通过 SQL 语句的执行过程，来了解 SQL 语句的执行效率。当一条 SQL 语句从客户端进程传递到服务器端进程后，Oracle 需要执行如下步骤：

（1）在共享池中搜索 SQL 语句是否已经存在。
（2）验证 SQL 语句的语法是否准确。
（3）执行数据字典来验证表和列的定义。
（4）获取对象的分析锁，以便在语句的分析过程中对象的定义不会改变。
（5）检查用户是否具有相应的操作权限。
（6）确定语句的最佳执行计划。
（7）将语句和执行方案保存到共享的 SQL 区。

从上面介绍的 SQL 语句的执行步骤可以发现，SQL 语句执行的前 4 步都是对 SQL 语句进行分析与编译，这需要花费很长的一段时间，显然 SQL 语句的编译是非常重要的，耗时的长短与 SQL 语句的结构清晰程度是密切相关的。

例如，查询 STUDENT 表中所有列的信息，该表含有 4 个字段，分别为 ID、NAME、AGE、SEX。首先使用 SET TIMING ON 语句显示执行时间，然后再使用“*”来替代所有的列名，执行语句和执行时间如下：

```
SQL> SET TIMING ON
SQL> SELECT * FROM student;
ID          NAME           AGE                SEX
--------- ------------- ---------------- -------------
1           马向林           22                 女
...
6           白雪             22                 女
已选择 6 行。
已用时间:  00: 00: 00.06
```

在 SELECT 子句中不使用“*”，而使用具体的列名，执行语句和执行时间如下：

```
SQL> SELECT id,name,age,sex FROM student;

ID           NAME           AGE                SEX
---------- -------------- ----------------- -----------------
1            马向林           22                 女
...
6            白雪             22                 女
已选择 6 行。
已用时间:  00: 00: 00.01
```

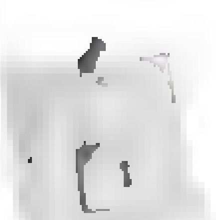
从执行结果可以看出，第二条 SQL 语句的执行时间小于第一条 SQL 语句的执行时间。这是因为 Oracle 系统需要通过数据字典将语句中的“*”转换成 STUDENT 表中的所有列名，然后再执行与第二条语句同样的查询操作，这自然要比直接使用列名花费更多的时间。



提示

如果再次执行这两条语句，会发现执行时间减少。这是因为所执行的语句被暂时保存在共享池中，Oracle 会重用已解析过的语句的执行计划和优化方案，因此执行时间也就减少了。

15.1.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1280-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16753-1-1.html

15.2 Oracle 中提交数据的问题

15.2.1 问题描述

我正在做毕业设计，再过几天就要答辩，但是数据库方面经常出问题，为什么我从 SQL*Plus 插入数据后，当时能够查询到刚刚插入的数据，但是关闭 SQL*Plus 之后，再打开时，就查询不到之前插入的数据了，这是为什么？

15.2.2 解决方法

数据没有提交前，只有执行 DML 语句的 ORACLE SESSION 可以看见刚插入的数据，你没有执行 COMMIT 就退出 SQL*Plus，则你执行的插入或修改被自动回滚，所以就看不到了。有两个方法可以解决此类情况的发生：

（1）执行数据插入后，使用 COMMIT 提交。

（2）在 SQL*Plus 中执行 SET AUTOCOMMIT ON 设置自动提交，这样就不会出现你遇见的问题了。

15.2.3 知识扩展——在确保完整性的情况下多用 COMMIT 语句

当用户执行 DML 操作后，如果不使用 COMMIT 命令进行提交，则 Oracle 会在回滚段中记录 DML 操作，以使用户使用 ROLLBACK 命令对数据进行恢复。Oracle 实现这种数据回滚功能，需要花费相应的时间与空间资源。所以，在确保数据完整性的情况下，尽量及时地使用 COMMIT 命令对 DML 操作进行提交。



使用 COMMIT 命令后，系统将释放回滚段上记录的 DML 操作信息、被程序语句获得的锁、redo log buffer 中的空间以及 Oracle 系统管理前面 3 种资源所需要的其他花费。

15.2.4 触类旁通



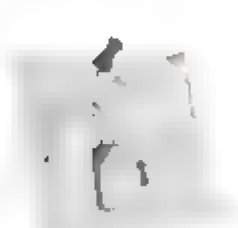
COMMIT 操作的时间问题。

网络课堂：http://bbs.itzcn.com/thread-16755-1-1.html

在每执行一条 DML 语句之后都需要使用 COMMIT 吗？那它到底都做了哪些工作？

COMMIT 通常是一个非常快的操作，而不论事务大小如何，你可能认为一个事务越大（换句话说，它影响的数据越多），COMMIT 需要的时间就越长。不是这样的，不论事务有多大，COMMIT 的响应时间一般都很“平”（flat，可以理解为无高低变化）。这是因为 COMMIT 并没有太多的工作去做，不过它所做的确实至关重要。这一点很重要，之所以要了解并掌握这个事实，原因之一是：这样你就能心无芥蒂地让事务有足够的大小。一种错误的概念认为分批提交可以节省稀有的系统资源，而实际上这只是增加了资源的使用。如果只在必要时才提交（即逻辑工作单元结束时），不仅能提高性能，还能减少对共享资源的竞争（日志文件、各种内部门等）。

15.2.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-3979-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16754-1-1.html

15.3 多次查询数据库的效率问题

15.3.1 问题描述

今天老师出了一道题目：高效率的查询出员工编号为 7566 和 7934 的员工信息，于是乎我编写了下面的两条 SQL 语句：

```
SQL> SELECT ename,sal FROM emp
  2  WHERE empno=7566;
SQL> SELECT ename,sal FROM emp
  2  WHERE empno=7934;
```

老师看过之后让我重新思考一下，我郁闷，这样编写不对吗？

15.3.2 解决方法

当执行每条 SQL 语句时，Oracle 在内部执行了许多工作：解析 SQL 语句、估算索引的利用率、绑定变量、读取数据块等等。由此可见，减少访问数据库的次数，实际上能提高 Oracle 的工作效率。因此，我们需要将上面的 SQL 语句改为：

```
SQL> SELECT a.empno,a.ename,a.sal,b.empno,b.ename,b.sal
  2  FROM emp a,emp b
  3  WHERE a.empno=7566
  4  AND
```

```
  5  b.empno=7934;

EMPNO       ENAME       SAL         EMPNO       ENAME       SAL
----------  ----------  ----------  ----------  ----------  ----------
7566        JONES       2975        7934        MILLER      1300
```

上述语句减少了访问数据库的次数，从而提高了效率。

15.3.3 知识扩展——减少表的查询次数

尽量减少表的查询次数，主要是指能使用一次查询获得数据，尽量不要去通过两次或更多次的查询获得。

例如，有两个表：学生信息表STUDENT和考试成绩表SCORE，现在需要获取马向林学生的考试成绩。我们可以采用多次查询的方式来获取：先从学生信息表中查询出马向林学生的学号（id），然后从考试成绩表中查询出该学生号所对应的考试成绩。如下：

```
SQL> SELECT stuid,result
  2  FROM score
  3  WHERE stuid=
  4  (SELECT id FROM student WHERE name='马向林');

   STUID     RESULT
---------- ----------
         1         89
已用时间:  00: 00: 00.04
```

如上述SQL语句，首先在WHERE子句中使用子查询语句从STUDENT表中获取姓名为马向林的学生号，将值传递给外部的WHERE子句，然后再由外部的SELECT语句从SCORE表中获取学号所对应的分数RESULT。这个过程中，使用了两次查询语句。

下面我们再使用连接查询，在表连接时必须选择连接顺序，将行较少的表连接到后面。例如，要连接3个相关表table1、table2和table3，假设表table1有10000行记录，表table2有1000行记录，表table3有100行记录。那么首先应该将table1连接到table2上，接着是table2连接到table3上。执行语句和执行时间如下：

```
SQL> SELECT stu.id,sc.result
  2  FROM student stu,score sc
  3  WHERE stu.id=sc.stuid AND stu.name='马向林';

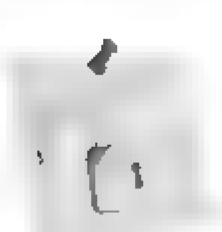
        ID     RESULT
---------- ----------
         1         89

已用时间:  00: 00: 00.01
```

如上述示例，通过表的连接查询方式同样可以检索出需要的数据，但是查询语句只使用

了一次，这就减少了对表的查询次数，从而执行时间比多次查询的时间要短。

15.3.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-3980-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16756-1-1.html

15.4 使用 HAVING 子句和使用 WHERE 子句的差异

15.4.1 问题描述

我分别使用 HAVING 和 WHERE 编写了两条功能相同的 SQL 语句，开启时间显示功能之后，得出的结果是两条 SQL 语句的执行时间竟然不相同，这应该如何理解呢？使用 HAVING 子句的 SQL 语句如下：

```
SQL> SELECT deptno,AVG(sal)
  2  FROM emp
  3  GROUP BY deptno
  4  HAVING deptno!=10;

    DEPTNO   AVG(SAL)
---------- ----------
        30 1566.66667
        20       2175

已用时间:  00: 00: 00.12
```

使用 WHERE 子句的 SQL 语句如下：

```
SQL>  SELECT deptno,AVG(sal)
  2   FROM emp
  3  WHERE ename!='JONES'
  4  GROUP BY deptno;

    DEPTNO   AVG(SAL)
---------- ----------
        30 1566.66667
        20       1975
        10 2916.66667

已用时间:  00: 00: 00.4
```

15.4.2 解决方法

使用 HAVING 子句只会在检索出所有记录之后才对结果集进行过滤，这个处理需要排序、总计等操作，如果能通过 WHERE 子句限制记录的数据，那就能减少这方面的开销了。因此使用 WHERE 子句的 SQL 语句执行时间要比使用 HAVING 子句的 SQL 语句执行时间短。

15.4.3 知识扩展——用 WHERE 替代 HAVING

在 SELECT 语句中，使用 GROUP BY 子句对行进行分组时，可以先使用 HAVING 对行进行过滤。例如对 SCOTT 模式下的 emp 表进行操作，根据 deptno 列进行分组，并且使用 HAVING 子句过滤 deptno 列的值为 20 的记录信息。如下：

```
SQL> SELECT deptno,COUNT(empno)
  2  FROM emp
  3  GROUP BY deptno
  4  HAVING deptno!=20;

    DEPTNO COUNT(EMPNO)
---------- ------------
        30            6
        10            3
已用时间:  00: 00: 00.06
```

同样也可以使用 WHERE 子句来替代 HAVING 子句的使用，实现同样的结果，但是执行时间却比较少，如下：

```
SQL> SELECT deptno,COUNT(empno)
  2  FROM emp
  3  WHERE deptno!=20
  4  GROUP BY deptno;

    DEPTNO COUNT(EMPNO)
---------- ------------
        30            6
        10            3
已用时间:  00: 00: 00.01
```

虽然 WHERE 子句与 HAVING 子句都可以用来过滤数据行，但 HAVING 子句会在检索出所有记录后才对结果集进行过滤；而使用 WHERE 子句就会减少这方面的开销。因此，一般的过滤条件应该尽量使用 WHERE 子句实现。



HAVING 子句一般用于对一些集合函数执行结果的过滤，如 COUNT()、AVG()等。除此之外，一般的检索条件应该写在 WHERE 子句中。

15.4.4 知识扩展——用 EXISTS 替代 IN

使用 IN 操作符用来检查一个值是否包含在列表中。EXISTS 与 IN 不同，EXISTS 只检查行的存在性，而 IN 检查实际的值。在子查询中，EXISTS 提供的性能通常比 IN 提供的性能要好。因此建议使用 EXISTS 操作符来替代 IN 操作符的使用，使用 NOT EXISTS 操作符替代 NOT IN 操作符的使用，来提高查询的执行效率。

例如查询本次考试中哪些学生参加了考试，也就是 SCORE 表中是否含有学生的编号。采用 IN 操作符，如下：

```
SQL>  SELECT stu.id,stu.name,stu.age,stu.sex,sc.result
  2   FROM score sc INNER JOIN student stu
  3   ON stu.id=sc.stuid
  4   WHERE sc.stuid IN
  5  (SELECT id FROM student);

ID          NAME            AGE         SEX              RESULT
---------- -------------- ---------- ---------------- -------
1           马向林          22          女               89
...
4           马林立          23          女               78
已用时间:  00: 00: 00.04
```

如上述示例，在判断哪些学生参加了考试时，采用了 IN 操作符的查询方式，这时，Oracle 会遍历查询学生表中所有学生的 id，判断考试成绩表中的 stuid 是否在这个集合中。

而如果采用 EXISTS 操作符，如下：

```
SQL>  SELECT id,name,age,sex
  2   FROM student
  3   WHERE EXISTS
  4   (SELECT 1 FROM score WHERE student.id=score.stuid);

ID              NAME                AGE               SEX
---------- -------------------- ---------------- -----------
1               马向林              22                女
2               殷国鹏              22                男
...
4               马林立              23                女
已用时间:  00: 00: 00.01
```

如上述示例，使用 EXISTS 操作符同样实现了查询效果。但是 EXISTS 的实现原理不一样，下面对它们进行简单比较：

- **IN**

确定给定的值是否与子查询或列表中的值相匹配。使用 IN 时，子查询先产生结果集，然后主查询再去结果集中寻找符合要求的字段列表，符合要求的输出，反之则不输出。

- **EXISTS**

指定一个子查询，检测行的存在。它不返回列表的值，只返回一个 TRUE 或 FALSE。其

运行方式是先运行主查询一次，再去子查询中查询与其对应的结果，如果子查询返回 TRUE 则输出，反之则不输出。再根据主查询中的每一行去子查询中查询。



由于 IN 操作符需要进行确切地比较，而 EXISTS 只需要验证存不存在，所以使用 IN 将会比使用 EXISTS 花费更多的查询成本，因此能使用 EXISTS 替代 IN 的地方，应该尽量使用 EXISTS。另外，尽量使用 NOT EXISTS 替代 NOT IN，使用 EXISTS 替代 DISTINCT。

15.4.5 触类旁通



如何使用 NOT EXISTS 替代 NOT IN？

网络课堂：http://bbs.itzcn.com/thread-16758-1-1.html

我要查询 STUDENT 表中 ID 不在 SCORE 表中的记录行，于是我编写了如下的 SQL 语句：

```
SQL> SELECT * FROM student
  2  WHERE id NOT IN
  3  (SELECT stuid FROM score);
```

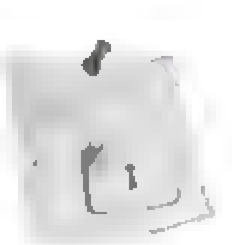
我从网上看到，使用 NOT EXISTS 替代 NOT IN 可以提高 Oracle 的执行效率，如何将上面 SQL 语句中的 NOT IN 替换 NOT EXISTS 呢？但是需要保证实现的功能不变。

NOT EXISTS 基本上可以完全替代 NOT IN 子句，因此在实际应用中能用 EXISTS 的 SQL 语句绝对不使用 IN。将上面 SQL 语句中的 NOT IN 子句修改为 NOT EXISTS 子句之后的完整语句如下：

```
SQL>  SELECT * FROM student
  2   WHERE NOT EXISTS
  3  (SELECT 1 FROM score WHERE score.stuid=student.id);
```

比较两条 SQL 语句执行时消耗的时间，使用 NOT IN 子句的 SQL 语句用时 3 分 25 秒，使用 NOT EXISTS 的 SQL 语句用时 12 秒。

15.4.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-1281-spid-35.html

视频教学：http://school.itzcn.com/video-vid-1282-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16757-1-1.html

15.5 查询的 SQL 语句返回量大时是否会走索引

15.5.1 问题描述

在 Oracle 中，如果查询的 SQL 语句返回量特别大，在执行的过程中 SQL 语句会不会走索引？如果不走索引的话会使 SQL 语句执行效率降低。

15.5.2 解决方法

是否会走索引，这需要看你的执行计划了。如果你的 Oracle 是 9i 以上的版本，可以通过在 SQL*Plus 中执行如下的 SQL 命令得到执行计划：

```
SET AUTOTRACE ON;
SET TIMEING ON;
```

执行你要执行的 SQL 语句就可以得到 SQL 语句的执行计划了。

15.5.3 知识扩展——使用索引的基本事项

通过索引查询要比全表扫描快得多，当 Oracle 找出执行查询的最佳路径时，Oracle 优化器就会使用索引。但是，在认识到索引的优点的同时，也要注意使用索引所需要付出的代价。索引需要占据存储空间，需要进行定期维护，每当表中有记录增减或索引列被修改时，索引本身也会被修改，这就意味着每条记录的 INSERT、DELETE、UPDATE 操作都要使用更多的磁盘 I/O。而且很多已经是不必要的索引，甚至会影响查询效率。



在实际应用中，对索引进行定期的重构相当必要。具体的重构操作请参考本书其他模式对象章节中与索引管理相关的内容。

所以在使用索引时，应该注意什么情况下使用它。一般，创建索引有如下几个基本原则：

- ❑ 对于经常以查询关键字为基础的表，并且该表中的数据行是均匀分布的。
- ❑ 以查询关键字为基础，表中的数据行随机排序。
- ❑ 表中包含的列数相对比较少。
- ❑ 表中的大多数查询都包含相对简单的 WHERE 子句。

除了需要知道什么情况下使用索引以外，还需要知道在创建索引时选择表中的哪些列作为索引列。一般，选择索引列有如下几个原则：

- ❑ 经常在 WHERE 子句中使用的列。
- ❑ 经常在表连接查询中用于表之间连接的列。
- ❑ 不宜将经常修改的列作为索引列。
- ❑ 不宜将经常在 WHERE 子句中使用，但与函数或操作符相结合的列作为索引列。
- ❑ 对于取值较少的列，应考虑建立位图索引，而不应该采用 B 树索引。



除了所查询的表没有索引，或者需要返回表中的所有行时，Oracle 会进行全表扫描以外，如果查询语句中带 LIKE 关键字，并使用了通配符“%”，或者对索引列使用了函数，Oracle 同样会对全表进行扫描。

15.5.4 触类旁通



为什么我写的 Oracle 语句都不肯走索引呢？

网络课堂：http://bbs.itzcn.com/thread-16760-1-1.html

在 STUDENT 表中，我设置了 ID 列为索引，当执行如下的语句时，竟然全表扫描然后显示出查询结果。怎么办啊？这样效率太低了！

```
SELECT COUNT(id) FROM student;;
```

你看看你设置的 STUDENT 表中的 ID 列是否有空值，或者是否没有加非空约束。如果索引列上有 NULL 值的话，它是不会走索引的，即使你强制它也不会走。你可以把 ID 列加一个非空约束应该就可以了。



Oracle 中索引使用问题！
网络课堂：http://bbs.itzcn.com/thread-16761-1-1.html

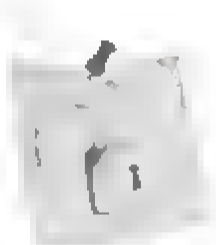
有个表 BOOK，包含有 3 个字段：BID、BNAME、PRICE，如果记录有 1000 条，并且为 BID 列创建了索引，怎么使用索引查询 BID 列的值以 2 开头的所有记录行？如 201、202……209。

可以编写如下的 SQL 语句来使用索引访问记录：

```
SELECT * FROM book
WHERE bid LIKE '2%';
```

“%”属于模糊匹配，此语句是可以用到索引的。但是如果是“LIKE ‘%2%’”时，就不会用到索引了。这是因为“%”出现在字符串之前时，需要匹配整个索引树内容，其代价比扫描还大，所以就不走索引了。

15.5.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-3981-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-17038-1-1.html

15.6 在索引中使用 IS NOT NULL 的问题

15.6.1 问题描述

表 DEPARTMENT 中的 DEPT_CODE 列为索引列，使用下面的语句对 DEPARTMENT 表查询时索引却失效了？为什么？

```
SQL> SELECT * FROM department WHERE dept_code IS NOT NULL;
```

15.6.2 解决方法

避免在索引中使用任何可以为空的列，Oracle 将无法使用该索引。对于单列索引，如果

列包含空值，索引中将不存在此记录，对于复合索引，如果每个列都为空，索引中同样不存在此记录。如果至少一个列不为空，则记录存在于索引中。例如：如果唯一性索引建立在表的 A 列和 B 列上，并且表中存在一条记录的 A、B 值为 123、NULL，Oracle 将不接收下一条具有相同 A、B 值为 123、NULL 的记录插入，然而如果所有的索引列都为空，Oracle 将认为整个键值为空，因此你可以插入 1000 条具有相同键值的记录，当然它们都是空，因为空值不存在于索引列中，所以 WHERE 子句中对索引列进行空值比较将使 Oracle 停用该索引。

如果你想要查询 DEPARTMENT 表中 DEPT_CODE 列的值不为空的记录行，则可以使用如下的 SQL 语句：

```
SQL> SELECT * FROM department WHERE dept_code>=0;
```

这样索引就有效了。

15.6.3 知识扩展——避免对索引列使用 NOT 关键字

在查询语句中，使用 NOT 关键字，可以为查询条件取反。例如，查询 SCOTT 模式下员工编号（empno，主键列）大于 7566 的员工信息，可以使用如下语句：

```
SQL> SELECT * FROM emp
  2  WHERE empno>7566;

EMPNO     ENAME      JOB          MGR      HIREDATE      SAL      COMM     DEPTNO
-------- ---------- ----------- ------- ------------ ------- -------- --------
7654        MARTIN   SALESMAN     7698     28-9 月-81    1250     1400     30
...
7934        MILLER   CLERK        7782     23-1 月-82    1300              10
已选择 10 行。
已用时间:  00: 00: 00.42
```

同样的，我们可以采用“NOT”关键字来实现，如下：

```
SQL> SELECT * FROM emp
  2  WHERE NOT empno<=7566;

EMPNO    ENAME        JOB          MGR HIREDATE             SAL      COMM    DEPTNO
-------- ------------ ------------ ---------- ---------- -------- -------- --------
7654      MARTIN      SALESMAN     7698 28-9 月 -81          1250     1400    30
...
7934      MILLER      CLERK        7782 23-1 月 -82          1300             10
已选择 10 行。
已用时间:  00: 00: 00.50
```

上述两条 SQL 语句实现的效果是一样的，但是，Oracle 执行上述两条语句所使用的方式是不一样的。由于 empno 列是主键列，所以该列上有索引，如果使用第一条语句，Oracle 会使用索引进行查询；而如果使用第二条语句，Oracle 会进行全表扫描。



在索引列上使用 NOT 关键字，与在索引列上使用函数一样，都会导致 Oracle 进行全表扫描。

实际上，索引的作用是快速地告诉用户在表中有什么数据，而不能用来告诉用户在表中没有什么数据。所以，类似的“!=”等操作符也应该避免在索引列上使用。

15.6.4 触类旁通



在索引列上进行计算是否会影响到执行效率？

网络课堂：http://bbs.itzcn.com/thread-16763-1-1.html

在 SCOTT 模式下的 DEPT 表中 DEPTNO 列为索引列，然后我编写了下面的 SQL 语句：

```
SELECT * FROM dept
WHERE deptno*12>240;
```

请问：这样会影响到 SQL 语句的执行效率吗？

在 WHERE 子句中，索引列不应该参与任何形式的计算，否则优化器将不使用索引而进行全表扫描，因此我们可以将上面的语句改写为：

```
SELECT * FROM dept
 WHERE deptno>240/12;
```

这样将会提高 SQL 语句的执行效率。



在索引上使用“>=”效率高还是使用“>”效率高？

网络课堂：http://bbs.itzcn.com/thread-16764-1-1.html

如果 DEPTNO 列上有一个索引，那么我是使用“>=”来对 DEPTNO 列进行比较的执行效率高呢？还是使用“>”号执行效率高？

在回答你的问题之前，我先使用“>=”来比较 DEPTNO，如下：

```
SQL> set timing on
SQL> SELECT empno,ename,sal FROM emp
  2  WHERE deptno>=20;

EMPNO      ENAME           SAL
---------- --------------- -----
7369       SMITH           800
7499       ALLEN           1600
...
7902       FORD            3000

已选择 11 行。
已用时间:  00: 00: 00.02
```

然后使用“>”来比较 DEPTNO，如下：

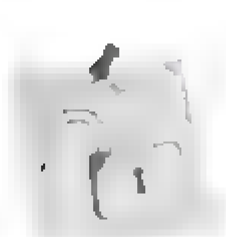
```
SQL> set timing on
SQL> SELECT empno,ename,sal FROM emp
  2  WHERE deptno>10;

EMPNO     ENAME            SAL
--------- ---------------- ----------
7369        SMITH            800
7499        ALLEN           1600
...
7902         FORD            3000
已选择 11 行。

已用时间:  00: 00: 00.06
```

上述两条 SQL 语句实现的功能是一样的，而执行所消耗的时间是不一样的，这是为什么呢？答案不难理解，使用“>=”的 SQL 语句在执行过程中，会直接跳到第一个 DEPTNO 等于 20 的记录。而使用“>”进行比较的 SQL 语句将首先定位到 DEPTNO 为 10 的记录，然后再扫描到第一个大于 DEPTNO 的记录，因此所消耗的时间要比使用“>=”比较索引所消耗的时间多。故使用“>=”比较索引的效率要高于使用“>”比较索引的效率。

15.6.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1284-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16762-1-1.html

15.7 多表连接性能很差吗

15.7.1 问题描述

今天我编写了一个多表连接的 SQL 语句，查询的数据量比较大，因此感觉查询速度有点慢。我听我同事说：多表连接性能太差，应该尽量减少多表连接操作，或者通过应用分步骤做。是这样的情况吗？多表连接性能真的会很差吗？

15.7.2 解决方法

首先请大家相信 Oracle 是关系数据库的鼻祖，多表连接是关系数据库的技术精髓之一。如果 Oracle 的多表连接都做不好，那就不叫关系数据库了。事实上，Oracle 提供了丰富的多表连接技术，关键是开发人员应该了解 Oracle 各种表连接技术的原理和适用“场景”，并结

合自已的应用有针对性地使用。一般只有在数据仓库系统中，才可能需要通过牺牲规范化设计来降低多表连接数量，提高多表连接性能。

本人曾经遇到过在一个银行的重大项目上，开发商把所有小表数据（主要是代码数据）全部下载到应用服务器上，目的就是要将数据库操作全部变成单表操作，通过应用程序实现表连接，保障系统性能。我马上指出这种方式的问题：首先应用服务器不是存储数据的地方，如果执意将数据下载到应用服务器，势必需要维护应用服务器与数据库服务器之间的数据同步和数据一致性。其次，本来一条简单的 SQL 就能实现的表连接功能，如果在应用程序中分步实现，人为增加应用复杂度。第三，应用程序与数据也紧耦合，改一个数据，你可能都需要修改程序。

说到底，还是开发人员不了解 Oracle 各种表连接技术的原理。针对这个项目应用情况，Oracle 其实会有效使用索引和嵌套循环连接的访问方式，保障高性能。在耐心说服客户相信 Oracle 表连接技术之后，该系统回到了简洁的开发方式，Oracle 多表连接效率非常高，目前系统运行地非常完美。

按照事物本身规律办事，返璞归真。千万别把问题复杂化！

15.7.3 知识扩展——选择 FROM 表的顺序

在 SELECT 语句的 FROM 子句中，可以指定多个表的名称。至于表与表之间的先后顺序，如果从查询结果来说，哪个表放在前面都一样，但是如果从查询效率来考虑，表之间的顺序是不能随意的。

一般来说，Oracle 的解析器在处理 FROM 子句中的表时，是按照从右到左的顺序，也就是说，FROM 子句中最后指定的表将被 Oracle 首先处理，Oracle 将它作为驱动表（Driving Table），并对该表的数据进行排序；之后再扫描倒数第二个表；最后将所有从第二个表中检索出来的记录与第一个表中的合适记录进行合并。

因此，建议在使用表的连接查询时，选择记录行数最少的表作为驱动表，也就是将它作为 FROM 子句中的最后一个表。例如，要使用两个相关表 table1 和 table2，其中，表 table1 有 10000 行记录，表 table2 有 1000 行记录。那么在 FROM 子句中，首先指定表 table1，接着是表 table2。

例如查询某个员工所在的部门名称及工作职位，需要连接查询 SCOTT 模式下的 emp 表和 dept 表，语句如下：

```
SQL>  SELECT empno,ename,sal,job,dname,dept.deptno
  2   FROM dept,emp
  3   WHERE emp.deptno=dept.deptno
  4   AND ename='MARTIN';

    EMPNO  ENAME         SAL        JOB          DNAME          DEPTNO
   ------  ----------    ------     ---------    -------------  ------
     7654  MARTIN        1250       SALESMAN     SALES            30

已用时间:  00: 00: 00.06
```

下面我们交换 emp 表与 dept 表的位置，再次进行查询，如下：

```
SQL> SELECT empno,ename,sal,job,dname,dept.deptno
  2  FROM emp,dept
  3  WHERE emp.deptno=dept.deptno
  4  AND ename='MARTIN';

    EMPNO   ENAME       SAL         JOB         NAME            DEPTNO
----------------------------------------------------------------------
     7654   MARTIN      1250        SALESMAN    SALES           30
已用时间:  00: 00: 00.01
```



emp 表中的记录数远远大于 dept 表中的记录数，所以两次 SELECT 语句的执行时间不同。

15.7.4 知识扩展——WHERE 子句的连接顺序

在执行查询的 WHERE 子句中，可以指定多个检索条件。Oracle 采用自右至左（自下向上）的顺序解析 WHERE 子句，根据这个顺序，表之间的连接应该写在其他 WHERE 条件之前，将可以过滤掉最大数量记录的条件写在 WHERE 子句的末尾。

例如，对 SCOTT 模式下的 emp 表进行操作，在 WHERE 子句中指定多个检索条件，执行语句和执行时间如下：

```
SQL> SELECT *
  2  FROM emp
  3  WHERE sal=3000
  4  AND
  5  (hiredate BETWEEN TO_DATE('1981-01-01','YYYY-MM-DD') AND TO_DATE
     ('1981-12-3 1','YYYY-MM-DD'));

EMPNO    ENAME     JOB        MGR    HIREDATE       SAL      COMM       DEPTNO
------------------------------------------------------------------------------
7902        FORD    ANALYST  7566   03-12 月-81   3000                 20
已用时间:  00: 00: 00.06
```

改变 WHERE 子句中检索条件的顺序，如下：

```
SQL>  SELECT *
  2   FROM emp
  3   WHERE
  4   hiredate BETWEEN TO DATE('1981-01-01','YYYY-MM DD') AND TO DATE
      ('1981-12-3 1','YYYY-MM-DD')
  5  AND
```

```
  6 sal 3000;

EMPNO    ENAME    JOB       MGR    HIREDATE       SAL    COMM    DEPTNO
-------- -------- --------- ------ -------------- ------ ------- ------
7902     FORD     ANALYST   7566   03-12月-81      3000            20
已用时间:  00: 00: 00.02
```

上述示例的 WHERE 子句中有两个检索条件，一条是检索入职时间为 1981 年份的，一条是检索工资为 3000 的，很明显工资为 3000 的检索条件可以过滤掉很大的记录数量，因此该条件应该放在 WHERE 子句中的末尾，提高效率。

15.7.5 知识扩展——使用表的别名

在使用表的连接查询时，经常用到表的别名，这个别名主要用来区别相同的表或者实现对表的简写。例如，连接 SCOTT 模式下的 emp 表和 dept 表，查询部门编号为 20 的所有员工姓名，如下：

```
SQL> SELECT e.ename
  2  FROM emp e INNER JOIN dept d
  3  ON e.deptno=d.deptno
  4  WHERE e.deptno=20;

ENAME
----------
SMITH
JONES
SCOTT
ADAMS
FORD
```

上面的查询语句中，为 emp 表命名别名为 e、为 dept 表命名别名为 d，这样就可以在使用列名的时候，使用表的别名来指定该列是哪个表中的，而不需要再完整地写出表的名称。

不过，并不是所有的列名都需要显式地指明它属于哪个表，只有当这个列名同时存在于多个表中时才需要，例如 dept 表中的 deptno 列与 emp 表中的 deptno 列，否则 Oracle 会出现解析错误。



虽然有些列只属于一个表，例如 ename 列只存在于 emp 表中，但是，如果在 SQL 语句中不指明它所属的表，那么这部分工作将会由 Oracle 自己去完成，这显然会增加 Oracle 的负担。所以，能够使用表的别名时就尽量使用。

15.7.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-1283-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16765-1-1.html

第16章　用户管理的备份与恢复

数据的备份和恢复是任何数据库都必不可少的功能之一。通过它可以保证数据的完整和正确性。数据的备份是指将数据以当前的状态存储副本形式，提供以后恢复使用。而数据的恢复正好和数据备份相反，它将数据以副本的形式恢复到数据库内。在实际的项目操作中，可能会因为某种原因导致数据库破坏或者数据丢失，为了能够确保数据的完整和安全性，可以使用数据的备份和恢复来解决这样的难题。

本章将介绍什么是数据库的备份与恢复，用户管理备份的不同形式，用户管理的完全恢复，以及用户管理的不完全恢复等内容。

16.1　Oracle数据库备份有哪些

16.1.1　问题描述

之前经营过一个网站，由于数据库出现故障导致数据库内大量的数据丢失，到现在尚未完全找回，直接影响到网站的运行。后来了解到数据库拥有备份和恢复机制，想通过使用这种机制将数据库进行备份，避免类似的事情重复出现，请问在Oracle数据库中数据库的备份有哪些？

16.1.2　解决方法

没错，对数据库的备份是十分重要的，特别是针对具有大量和珍贵数据的数据库。其实在数据库中数据库备份就是将数据复制一份保存起来。通常Oracle数据库备份有两种：物理备份和逻辑备份。

- **物理备份**

物理备份就是将数据库的数据文件复制一份保存，其中物理备份又分为脱机和联机两种不同的备份方式。

- **逻辑备份**

逻辑备份就是将数据通过使用Oracle导出工具将其导出备份，它和物理备份的概念是不相同的。

表16-1列出了使用两种不同的数据备份方式的优点、缺点以及最佳的备份时机。

表 16-1　逻辑备份和物理备份的优缺点

	逻辑备份（导出方式）	[illegible]	
		脱机备份	联机备份
优点	能够针对行对象进行备份；能够通过跨平台实施备份并迁移数据，而不需要关闭数据库	备份和恢复迅速，容易达到低维护，高安全性，执行效率高	备份的时间短；备份时数据库仍可使用；可达到秒级恢复（恢复到某一时间点上）；对几乎所有数据库对象都可以实现恢复
缺点	导出方式并不能保证介质失效，仅仅是逻辑上的备份	单独使用时，只能提供到某一时间点的恢复，不能按表和用户恢复，而且必须关闭数据库	实现过程比较复杂；需要较大的空间存放归档文件；操作时不允许失误，否则恢复不能进行
使用时机	一般用于有规律的日常备份	数据库可以暂时关闭，或者需要和联机备份配合使用时	数据访问量小，或者需要实现表空间或者数据库文件级的备份，或者需要更高的精确备份时

16.1.3　知识扩展——数据库备份概述

物理备份和逻辑备份在不同的环境效果也不相同，下面是对两种不同的备份方式详细的介绍。

1．物理备份

物理备份是通过计算机将数据文件、日志文件和控制文件等，进行复制另外存储，这样就做到了物理备份。这样的备份可以实现据库的完整恢复，但是必须运行在归档模式下。

在物理备份中又分为完全数据库脱机备份、部分数据库脱机备份和部分数据库联机备份。

❑ **完全数据库脱机备份**

数据库脱机表示在数据库正常关闭的情况下进行备份。而完全数据库脱机备份就是在关闭数据库时将所有的数据文件、日志文件以及控制文件进行备份。在备份时需要使用 SHUTDOWN 命令的 NORMAL、IMMEDIATE 或 TRANSACTIONAL 选项来完成这项操作。


提示

如果数据库没有关闭就进行脱机备份，那么数据库文件或者其他的文件将会被占用，因此就无法正常地完成备份。

❑ **部分数据库脱机备份**

部分数据库脱机备份是指，通过使用语句将数据库内部分表空间、表或者数据库进行脱机备份，它可以在数据库脱机进行也可以在联机时进行。因为该备份方式只对脱机的部分进行备份操作。

❑ **部分数据库联机备份**

该备份方式可以在数据链接时进行备份操作，它可以将数据库中的部分表空间、控制文件、数据文件和归档日志文件进行备份，和完全数据库脱机备份相比减少了大量的工作量。

2．逻辑备份

在数据库中进行逻辑备份就是把数据库中的一组数据记录写入到一个文件中，而这个文

件与它们存放的物理位置没有任何关系。

在 Oracle 数据库中，进行逻辑备份使用 Data Pump Export（数据泵导出）来完成。同时为了程序的需要可以使用 Data Pump Import（数据泵导入）将数据进行恢复。

❑ **Data Pump Export**

Data Pump Export 是将数据库、用户、表空间或者表以文件的形式写入到一个 XML 文件中。



在使用 Data Pump Export 实用程序导出的过程中，可以选择是否导出与数据表相关的数据字典信息，例如授权、索引和约束条件等。

❑ **Data Pump Import**

Data Pump Import 是将数据进行导入的操作。Data Pump Import 将会把 Data Pump Export 导出的文件进行执行操作。该文件包含一个 CREATE TABLE 命令创建一个表，然后使用 INSERT 语句将数据插入到表中。



在使用 Data Pump Import 和 Data Pump Export 程序时，可以进行远程的操作，减少用户间接操作系统文件。

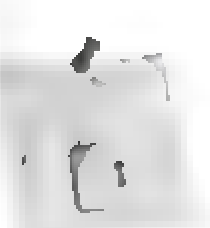
16.1.4 知识扩展——数据库备份命令

在 Oracle 数据库中，由于备份的文件不相同，因此备份时使用到的命令也不相同，在表 16-2 中列出了备份命令。

表 16-2 文件类型和备份命令

文件类型	备份命令	示例
数据文件	操作系统命令	COPY c:\datafile1.dbf d:\datafile1.dbf，表示将目录 c:\的 datafile1.dbf 文件复制到 d:\目录下
日志文件	操作系统命令	COPY c:\logfile1.log d:\logfile1.log，表示将目录 c:\的 logfile1.log 文件复制到目录 d:\下
控制文件	SQL 命令	ALTER DATABASE BACKUP CONTROLFILE TO confile1.ctl
初始化参数文件	SQL 命令	CREATE PFILE SIDinit.ora FROM SPFLE
数据库逻辑对象（表、索引等）	Export 命令	Export system/password

16.1.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1285-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1286-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16864-1-1.html

16.2 如何实现 Oracle 完全数据库脱机备份

16.2.1 问题描述

在对数据库进行备份时，经常会使用到完全数据库脱机备份，将数据库中的数据文件、日志文件和控制文件等进行备份，那么在 Oracle 中如何实现这样的操作的？

16.2.2 解决方法

实现 Oracle 完全数据库脱机备份非常简单，首先用户必须以数据库管理员的身份登录到系统中。然后关闭数据库，即数据库在脱机状态。接着将数据库通过 COPY 关键字复制一份，最后启动数据库。详细代码如下。

```
SQL> CONNECT sys/admin AS sysdba;
已连接。

SQL> SHUTDOWN IMMEDIATE;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。

SQL> HOST COPY F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF BACKUP;
已复制        1 个文件。

SQL> HOST COPY F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG BACKUP;
已复制        1 个文件。

SQL> HOST COPY F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL BACKUP;
已复制        1 个文件。

SQL> STARTUP;
ORACLE 例程已经启动。
Total System Global Area  431038464 bytes
Fixed Size                  1375088 bytes
Variable Size             327156880 bytes
Database Buffers           96468992 bytes
Redo Buffers                6037504 bytes
数据库装载完毕。
数据库已经打开。
```

在代码中，使用 sys 用户登录。其中 SHUTDOWN IMMEDIATE;命令用于关闭数据库。接着使用 COPY 进行备份，最后使用 STARTUP 关键字打开数据库。

16.2.3 知识扩展——完全数据库脱机备份

在对数据库进行完全数据库脱机备份时，主要是对数据文件、日志文件和控制器等文件通过使用系统命令进行复制备份，而这些任务必须是在数据库关闭脱机状态下来完成的。在进行备份时，备份的管理员必须拥有相应的权限。

在备份数据文件、日志文件和控制文件时必须知道这些文件的存放地址，那么就需要通过查询数据字典来获取要被备份的文件地址。

1．数据文件

要获取当前数据库的数据文件存放地址需要查询数据字典视图 DBA_DATA_FILES，详细代码如下。

```
SQL> SELECT file name FROM dba data files;

FILE NAME
-----------------------------------------------
F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF
F:\APP\ADMINISTRATOR\ORADATA\ORCL\UNDOTBS01.DBF
F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSAUX01.DBF
F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSTEM01.DBF
F:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF
F:\APP\ADMINISTRATOR\ORADATA\ORCL\TEST.DBF
F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF
F:\APP\ADMINISTRATOR\ORADATA\ORCL\TESTUSERS.DBF

已选择 8 行。
```

在代码中，通过使用 SELECT 语句查询 DBA_DATA_FILES 视图字典，获取了 8 条数据文件存放位置。

2．日志文件

日志文件也需要通过查询来获取数据，在这里需要查询数据字典视图 V$LOGFILE。详细代码如下。

```
SQL> SELECT member FROM v$logfile;

MEMBER
--------------------------------------------

F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
F:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
```

在这段代码中，获取到的数据是日志文件的存放地址，在对日志文件进行备份时需要使用这里的地址信息。

3．控制文件

控制文件的地址信息存储在 V$CONTROLFILE 数据字典视图中，查询该数据字典即可获取到控制文件的存放信息，详细代码如下。

```
SQL> SELECT name FROM v$controlfile;

NAME
------------------------------------------------------------

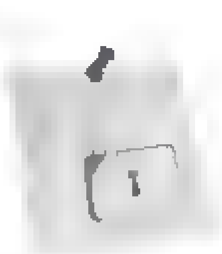
F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL
F:\APP\ADMINISTRATOR\FLASH_RECOVERY_AREA\ORCL\CONTROL02.CTL
```

到这里已经成功地获取到了数据文件、日志文件和控制文件的存放地址信息，接下来就是进行数据的备份操作，备份时需要使用到以下语法，代码如下。

```
SHUTDOWN IMMEDIATE
HOST COPY file BACKUP
STARTUP
```

在语法中有 3 行语句，其中第一行 SHUTDOWN IMMEDIATE 表示将数据库关闭处于脱机状态。第二行就是进行数据备份操作，其中 file 表示备份的文件地址，在这里填写上前面查询出来的数据文件、日志文件和控制文件的存放地址即可。最后一行是打开数据库，将数据库设置为初始状态。

16.2.4　网络课堂



视频教学：http://school.itzcn.com/video-vid-1287-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16865-1-1.html

16.3　为什么使用部分数据库脱机备份

16.3.1　问题描述

刚接触数据库，对数据库的备份不太了解。在编写代码对数据库进行备份时，几个数据库管理的前辈要求使用部分数据脱机备份。因为在前面学习过完全数据脱机备份。对部分数据脱机备份的概念和优点不太了解，为什么前辈要求尽量使用部分数据脱机备份，而不使用完全数据脱机备份呢？

16.3.2　解决方法

部分数据脱机备份，从词的意义上理解就是将数据内部分文件或者表空间进行备份。在

备份期间没有被列入备份的表空间还可以正常使用。可能数据库管理员知道有的数据库能正在使用，如果将其设置为脱机状态，那么其他用户无法访问该数据文件。

16.3.3 知识扩展——部分数据库脱机备份

在对部分数据库脱机备份时，不能将 SYSTEM 表空间，或者任何包含有活动回退段的表空间设置为脱机状态，因为，这些表空间也就不能在脱机状态下进行备份。



注意　由于 SYSTEM 表空间包含了数据字典，而数据字典存储了与数据库对象有关的所有信息。如果 SYSTEM 表空间脱机，那么将无法识别任何数据库对象，如表、索引等。

在执行部分数据库脱机备份前，同样需要查看该数据库的存储位置获取相应的信息。然后将其备份的表空间进行脱机操作，其语法如下。

```
ALTER TABLESPACE tablespace_name OFFLINE | ONLINE
```

上述语法中，tablespace_name 表示将要被脱机的表空间名称；OFFLINE 关键字则是该表空间设置为脱机状态；ONLINE 表示联机状态。通常脱机备份完成之后将其设置为联机状态。

将需要备份的部分表空间脱机之后就可以进行备份操作，下面代码是在 DOS 命令行，语法如下所示。

```
tablespace_file COPY tablespace_name out_tablespace_file
```

语法中，tablespace_file 表示要备份的数据库路径；tablespace_name 表示备份的数据库名称；out_tablespace_file 表示存放备份好的数据文件的路径。

在下面步骤中详细描述了如何实现部分数据库脱机备份的实例代码。

（1）在对表空间进行备份之前，首先需要通过查看数据字典视图 DBA_DATA_FILES 来获取需要备份表空间数据文件存放的位置，详细代码如下。

```
SQL> SELECT tablespace name,file name
  2  FROM dba data files
  3  WHERE tablespace_name='TEST';
TABLESPACE NAME         FILE NAME
------------------      ------------------
FILE_NAME               F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF
```

（2）获取到表空间数据文件位置后，将该表空间设置为脱机状态，详细代码如下。

```
SQL> ALTER TABLESPACE users OFFLINE;
表空间已更改。
```

（3）接下来开始对数据库表空间数据文件进行脱机备份，前面通过使用命令获取了数据库表空间文件路径，使用 COPY 命令将该文件备份到 D:\MyDB 目录下，执行代码如下。

```
F:\APP\ADMINISTRATOR\ORADATA\ORCL\> copy USERS01.DBF D:\MyDB\;
```

```
已复制              1 个文件
```

上述代码是在运行 CMD 命令行中执行的代码，将 USERS01.DBF 复制到 D:\MyDB\目录下。

（4）复制完成后将数据库表空间设置为联机状态，需要使用 ONLINE 命令，详细代码如下。

```
SQL> ALTER TABLESPACE users ONLINE;
表空间已更改。
```

16.3.4 知识扩展——部分数据库联机备份

部分数据库联机备份和部分数据库脱机备份的工作原理是几乎相似的，唯一不相同的是它们备份的数据库状态不相同，一个是在联机时进行备份另外一个则是在脱机状态下进行的备份操作。

由于在联机状态下备份数据库，用户还可以进行访问数据库，甚至还可以对数据库数据进行增、删、改、查等操作，因此使得数据库文件之间存在不同步。当对备份的数据进行恢复到数据库时，可以在归档模式下进行。

首先对当前的数据库的日志模式切换为归档模式，切换代码如下。

```
SQL> ALTER DATABASE ARCHIVELOG;
数据库已更改。
```

然后使用 ARCHIVE LOG LIST 语句，检测当前数据库的日志模式是否已经切换为归档模式，代码如下。

```
SQL> ARCHIVE LOG LIST;
数据库日志模式          存档模式
自动存档                启用
存档终点                USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列      24
下一个存档日志序列      26
当前日志序列            26
```

在运行结果中，如果数据库日志模式选项为存档模式，那么就表明已经成功地将数据库日志文件设置为了归档模式。

然后使用 BEGIN BACKUP 语句将数据库表空间设置为备份状态，设置语法如下所示。

```
ALTER TABLESPACE tablespace_name BEGIN BACKUP
```

接着将其数据库进行备份操作，在这里的备份操作和前面的脱机备份操作相同，使用 COPY 命令来完成。语法如下所示。

```
tablespace_file COPY tablespace_name out tablespace_file
```

最后则是将当前操作的表空间结束备份状态，恢复正常，在这里需要使用 END BACKUP

命令。

```
ALTER TABLESPACE tablespace name END BACKUP
```

16.3.5 知识扩展——备份控制文件

在 Oracle 数据库中，数据库的结构信息通常存储在控制文件中，如果控制文件损坏那么对数据库的损失也是很明显的。因此对数据库控制文件的备份显然非常必要的。

备份控制文件有 3 种不同的备份方式，下面就是 3 种控制文件的备份方式：

1．使用系统 COPY 命令

使用 COPY 命令备份控制文件和 16.3.3 讲解的使用 COPY 对数据库进行脱机备份的步骤相同，在这里不再多讲。但是在备份时数据库必须处于关闭状态。

2．使用 ALTER DATABASE BACKUP CONTROLFILE TO 语句

使用该 ALTER DATABASE BACKUP CONTROLFILE TO 语句可以将控制文件存储到二进制文件中，使用时其语法如下所示。

```
ALTER DATABASE BACKUP CONTROLFILE TO location
```

其中语法中 location 表示控制文件备份的位置。具体使用该语句备份控制文件代码如下。

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO 'D:\MyDB\mycontrol.ctl';
数据库已更改。
```

备份完成之后在 D:\MyDB\目录下将会生成名称为 mycontrol.ctl 的二进制文件。

3．使用 ALTER DATABASE BACKUP CONTROLFILE TO TRACE 语句

通过使用 ALTER DATABASE BACKUP CONTROLFILE TO TRACE 语句可以将控制文件备份到跟踪文件中，其语法如下所示。

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

该语句具体实现代码如下。

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
数据库已更改。
```

备份完成后，用户可以通过以下代码来查看备份文件的位置可以使用系统参数 USER_DUMP_DEST 来完成，详细代码如下。

```
SQL> SHOW PARAMETER USER DUMP DEST;

NAME               TYPE      VALUE
------------------ --------- ----------------------------------------------
user_dump_dest     string    f:\app\administrator\diag\rdbms\orcl\orcl\trace
```

16.3.6 知识扩展——验证备份数据

在用户对数据库文件进行备份后，为了确保数据备份文件的有效性，可以使用 DBVERIFY 工具对备份文件进行检验。

在 Oracle 数据库中 DBVERIFY 工具和 Oracle 服务软件一起安装的工具，通过这个工具可以检测备份的数据文件是否有效。检验时需要使用 DBVERIFY 工具命令 DBV 命令。表 16-3 就列出了常用的 DBV 命令的一些参数。

表 16-3　常用 DBV 命令参数

参数	说明
FILE	进行验证的数据文件路径和名称
START	进行验证的起始数据块位置；如果不指定 START 参数，默认将对数据文件中的所有数据块进行验证
END	进行验证时终止数据块的位置，利用 START 和 END 参数，可以对数据文件中指定位置的数据块进行验证
BLOCKSIZE	指定数据文件中数据块的大小
LOGFILE	指定一个作为验证结果的输出文件。如果未指定该参数，验证结果将直接显示在用户终端上
FEEDBACK	该参数的作用是动态显示验证的进度，如果将它设置为 n，那么 DBVERIFY 在每验证 n 个数据块后显示一个字符“.”，这样表示验证完成情况的进度栏
PARFILE	指定一个包含参数的设置文件

在表中列出了 DBV 命令的参数，用户可以根据自己的需求选择不同的参数，不过有的参数是必须存在的，例如 FILE 等。

例如下面实例步骤中，就是对备份好的 USER.DBF 文件进行验证，是否该文件可以正确地执行而没有损坏。

首先用户打开一个操作系统的命令提示窗口，在命令提示窗口中输入 DBV 命令进行检测，如果备份文件有问题，那么在提示窗口内有相应的信息提示。代码如下。

```
C:\>DBV FILE=F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF BLOCKSIZE=8192
DBVERIFY: Release 11.2.0.1.0 - Production on 星期四 8月 4 11:09:17 2011
Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.
DBVERIFY - 开始验证: FILE = F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF
DBVERIFY - 验证完成
检查的页总数: 12800
处理的页总数 (数据): 0
失败的页总数 (数据): 0
处理的页总数 (索引): 0
失败的页总数 (索引): 0
处理的页总数 (其他): 130
处理的总页数 (段)  : 0
失败的总页数 (段)  : 0
空的页总数: 12670
```

```
标记为损坏的总页数: 0
流入的页总数: 0
加密的总页数: 0
最高块 SCN: 1088765 (0.1088765)
```

在提示信息中，页表示 Oracle 中的数据块，如果失败的页总数为 0，表明此备份文件完好无损，如果备份文件不为 0，那么表明备份的文件有所损坏。

16.3.7 触类旁通



在对数据库进行备份时，关闭数据库被拒绝？

网络课堂：http://bbs.itzcn.com/thread-16867-1-1.html

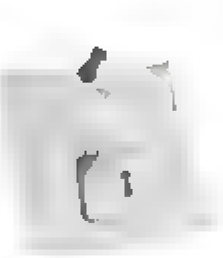
在对数据库做联机备份时，首先将数据库日志设置为归档模式。然后将需要备份的表空间设置为备份状态，可是此时需要关闭数据库却被拒绝，运行代码如下。如何解决这样的问题。

```
SQL> ALTER TABLESPACE users BEGIN BACKUP;
表空间已更改。

SQL> SHUTDOWN IMMEDIATE;
ORA-01149: 无法关闭 - 文件 4 设置了联机备份
ORA-01110: 数据文件 4: 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF'
```

当把表空间设置为备份状态后，该表空间数据文件的检验点号将停止修改，并对数据文件做联机备份标记。因此就会拒绝关闭数据库，如果此时发生停电或者其他原因导致强制关闭数据库，那么数据库就不能被启动，此时的数据库文件是不同步的。

16.3.8 网络课堂



视频教学：http://school.itzcn.com/video-vid-1288-spid-35.html

视频教学：http://school.itzcn.com/video-vid-1289-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16866-1-1.html

16.4 数据文件损坏怎么办

16.4.1 问题描述

在使用 Oracle 数据库进行对数据备份时，使用的是完全脱机备份方式，首先需要对数据库进行关闭，执行 SHUTDOWN IMMEDIATE 语句关闭数据库。然后进行脱机备份，当我把数据库备份好之后，准备使用 STARTUP 语句启动数据库时却出现了异常，异常代码如下，

大家帮我解决下这个问题。

```
SQL> STARTUP;
ORACLE 例程已经启动。

Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              327156880   bytes
Database Buffers           96468992    bytes
Redo Buffers               6037504     bytes
数据库装载完毕。
ORA-01157: 无法标识/锁定数据文件 7 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 7: 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF'
```

在出现该异常之前，是对 USERS 数据库进行脱机备份的，备份好之后就出现这样的异常信息。

16.4.2 解决方法

在异常信息中可以看出，USERS.DBF 数据文件丢失或者损坏。那么此时需要对数据文件信息进行恢复，但是恢复时不可以单单就恢复损坏或者丢失的数据文件，这样会造成数据不一致还是无法启动数据库的。只有对所有的数据文件进行恢复才可以。详细恢复代码如下。

```
SQL> HOST COPY F:\OracleDB USERS.DBF F:\APP\ADMINISTRATOR\ORADATA\
ORCL\USERS.DBF
```

上述代码是把存放在 F:\OracleDB 文件夹下的 USERS.DBF 数据文件恢复数据库的目录下。到这里就已经将损坏的数据文件进行了恢复，然后通过使用 STARTUP 命令再次打开数据库。

```
SQL> STARTUP;
ORACLE 例程已经启动。

Total System Global Area  431038464 bytes
Fixed Size                  1375088 bytes
Variable Size             327156880 bytes
Database Buffers           96468992 bytes
Redo Buffers                6037504 bytes
数据库装载完毕。
ORA-01113: 文件 7 需要介质恢复
ORA-01110: 数据文件 7: 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF'
```

出现这样的异常是因为，在恢复数据库时，如果只恢复单个文件不能完全将数据库进行恢复，需要将数据库进行介质恢复。

这是由于数据库认为这个数据文件遭到破坏了，需要使用 RECOVER 命令通过备份、日

志信息来恢复。数据库的备份恢复是个比较复杂的问题，但是这个实例的解决办法还是比较简单的。

```
RECOVER DATAFILE 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF'
完成介质恢复。
```

16.4.3　知识扩展——Oracle 数据库完全恢复机制

在操作数据库时，如果数据库文件发生介质失败时，可以通过使用命令将备份好的数据文件转存到数据库目录中，将数据文件恢复到发生错误之前。

对数据库进行完全恢复需要做以下两点，首先当数据库发生介质错误时，使用数据备份文件对损失的数据进行修复，其次就是当把数据文件修复完成后，由于新修复的文件和其他数据文件之间数据不同步，所以还是无法启动数据库。那么此时就需要通过 SQL 语句进行归档日志对数据库进行恢复。

在图 16-1 中是通过使用 Oracle 数据库机制对 USERS 数据库进行完全恢复的过程。

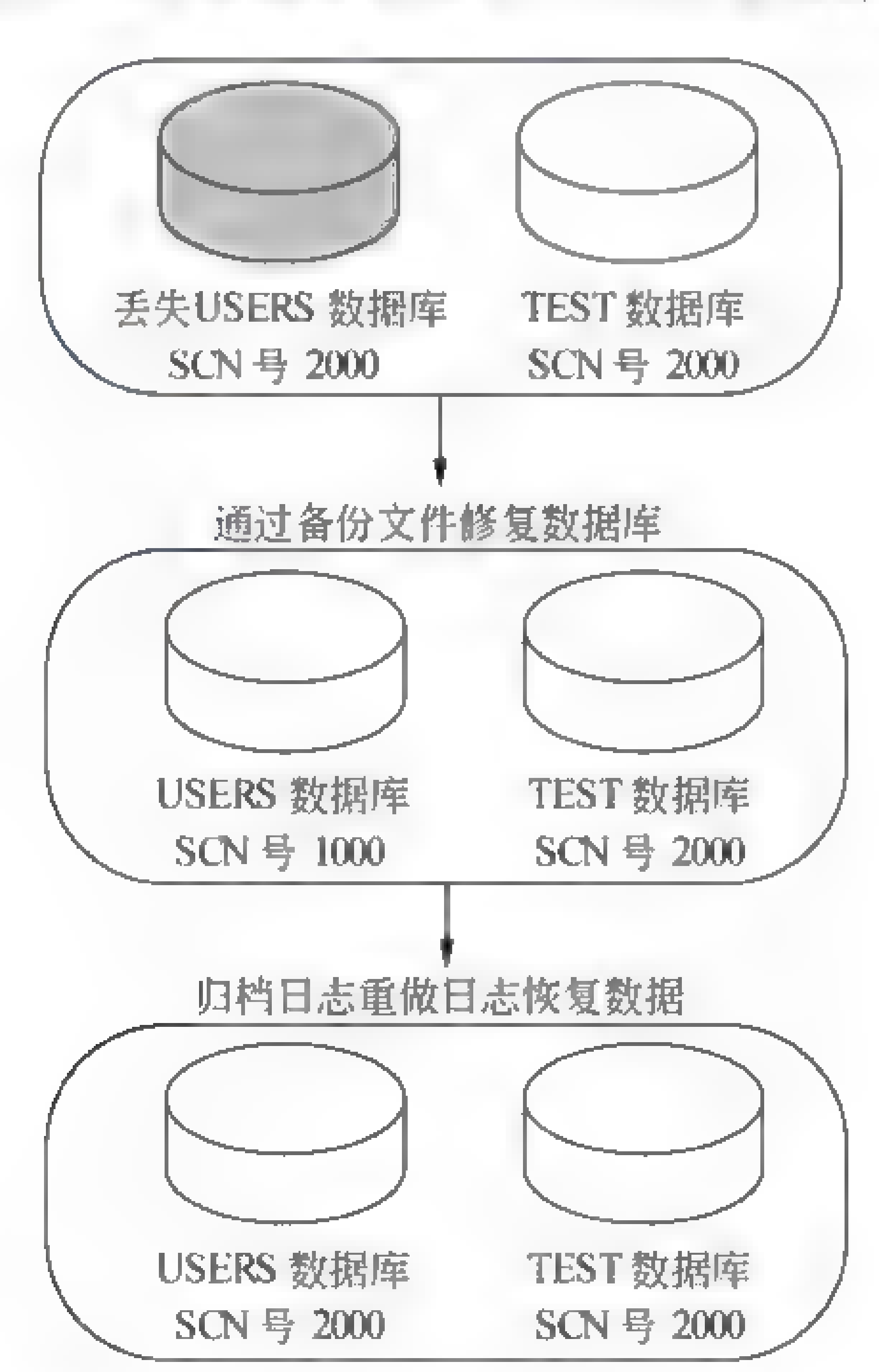


图 16-1　USERS 数据库完全恢复过程

在上图中，USERS 数据库丢失，需要通过使用备份文件对数据库进行恢复。恢复后数据库的 SCN 号与其他文件的 SCN 号不符合，因此还是无法打开数据库。那么此时需要使用归档日志恢复，将数据库的 SCN 号修改成与其他文件 SCN 号相同。

在恢复数据库时需要使用以下三种命令对数据库进行完全恢复。

- **RECOVER DATABASE**

用于恢复数据库的多个数据文件，该命令只能在 MOUNT 状态下使用。

- **RECOVER TABLESPACE**

用于恢复一个或多个表空间的所有数据文件，该命令只能在 OPEN 状态下运行。

- **RECOVER DATAFILE**

用于恢复一个或多个数据文件，该命令可以在 MOUNT 状态和 OPEN 状态下运行。同时，可以指定数据文件的名称和数据文件的编号。

16.4.4 知识扩展——非归档模式下的数据库恢复

有时数据库可能处于非归档模式，此时数据库文件被损坏无法启动时就需要通过使用非归档模式对数据库进行恢复操作。

非归档模式下恢复数据库文件非常简单，只需要数据库管理员将备份好的数据库文件覆盖到数据库内即可。

对非归档模式数据库进行恢复需要四个步骤：第一步关闭数据库，对损坏数据库文件进行修复；第二步启动数据库测试是否恢复成功，如果出现需要介质恢复，说明单个数据文件不能完全恢复数据库；第三步将备份的所有数据库文件都进行恢复；第四步重新打开数据库。



注意：如果恢复的数据库在非归档模式下，损坏的文件只有一个，那么也必须将所有的备份文件进行恢复。因为如果只恢复一个文件数据可能不一致。

当数据库文件受损，需要进行数据库恢复，如果只针对损坏文件进行恢复可能造成数据不同步，只有对所有数据文件进行恢复才可以达到恢复的效果，详细步骤如下所示。

（1）关闭数据库，对备份文件中的 USERS.DBF 复制到数据库目录中，代码如下。

```
SQL>SHUTDOWN IMMEDIATE;
ORACLE 例程已经关闭。
SQL>HOST COPY D:\MyDB\USERS.DBF F:\app\administrator\oradata\
orcl\USERS.DBF;
已复制         1 个文件
```

（2）启动数据库，查看文件是否恢复成功，详细代码如下。

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              348128400   bytes
Database Buffers           75497472    bytes
Redo Buffers               6037504     bytes
数据加载完毕。
ORA-01113: 文件 4 需要介质恢复
ORA-01110: 数据文件 4: F:\app\administrator\oradata\orcl\USERS.DBF
```

因为修复的数据文件与其他文件不同步，所以在恢复数据文件时只恢复单个数据文件是不能完全将数据进行恢复的。

（3）为了能够将数据库完全地恢复，通过使用命令将脱机备份的所有数据库文件进行恢复，代码如下。

```
SQL>HOST COPY D:\MyDB\USERS.DBF F:\app\administrator\oradata\orcl\
REDO02.LOG;
已复制         1 个文件
SQL>HOST COPY D:\MyDB\USERS.DBF F:\app\administrator\oradata\orcl\
CONTROL01.CTL;
已复制         1 个文件
```

（4）此时再次使用 STARTUP 命令打开数据库，代码如下。

```
SQL> startup
ORACLE 例程已经启动。

Total System Global Area    431038464   bytes
Fixed Size                  1375088     bytes
Variable Size               348128400   bytes
Database Buffers            75497472    bytes
Redo Buffers                6037504     bytes
数据加载完毕。
```

上述代码运行结果可以看出已经成功的启动了数据库。表明已经成功地将数据库进行了恢复。


注意

当数据库处于非归档模式时，执行脱机备份之后所有的更改都将会丢失，而这些更改不能恢复只能重新再次操作。

16.4.5 知识扩展——归档模式下的数据库恢复

恢复归档模式下的数据库文件是将备份的数据文件的副本放在数据库文件夹中，然后重新启动数据库，这将启动实例并且打开控制文件，然后使用 RECOVER 命令，并重新应用归档的重做日志中的事务。

当数据库处于归档模式时，并且在关闭状态下数据库发生介质损坏时，那么在打开数据库时，后台 DBWR 进程会将错误信息写入到跟踪文件中，并输出 ORA-01157 和 ORA-01110 错误提示信息，详细代码如下。

```
ORA-01157: 无法标识/锁定数据文件 4 - 请阅读 DBWR 跟踪文件
ORA-01110: 数据文件 4: F:\app\administrator\oradata\orcl\USERS.DBF
```



当数据文件发生介质损坏时，在执行 SQL 命令恢复数据库之前必须通过操作命令修复数据文件。如果数据文件丢失，那么只需要将备份文件放回原位置即可。

如果是数据文件所在的磁盘损坏，那么只需要将数据文件复制到其他磁盘，然后更改控制文件，重新定位数据文件。

当数据库处于 MOUNT 状态时，数据库管理员可以改变任何数据文件的位置，但主要改变 SYSTEM 表空间数据文件的位置，例如下属代码。

```
SQL> ALTER DATABASE RENAME FILE 'F:\app\administrator\oradata\orcl\
USERS.DBF'
  2  TO 'D:\MyDB\USERS.DBF'
```

当数据库处于 OPEN 状态时，数据库管理员可以更改除 SYSTEM 表空间之外的数据库文件的位置。例如以下实例代码。

```
SQL>ALTER DATABASE DATAFILE 'F:\app\administrator\oradata\orcl\USERS.DBF'
OFFLINE;
SQL>ALTER TABLESPACE USERS RENAME
  2  DATAFILE 'F:\app\administrator\oradata\orcl\USERS.DBF'
  3  TO 'D:\MyDB\USERS.DBF'
```

16.4.6 触类旁通



USERS 表空间损坏，如何恢复？

网络课堂：http://bbs.itzcn.com/thread-16870-1-1.html

在 USERS 表空间中存储了大量的重要信息，为了数据库的安全我经常对该表空间进行联机备份，可是今天数据库以外的发生了故障，数据库被强制关闭数据文件损坏。那么如何修复该表空间。以下代码是对 USERS 表空间的备份代码。

```
SQL> CONNECT sys/admin AS sysdba;
已连接。

SQL> ALTER TABLESPACE users BEGIN BACKUP;
表空间已更改。

SQL> HOST COPY F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF F:\OracleDB
\USERS.DBF
已复制         1 个文件。

SQL> ALTER TABLESPACE users END BACKUP;
表空间已更改。
```

代码中，使用数据库管理员身份登录，把 USERS 表空间设置为备份状态将正确的 USERS 表空间备份文件存放在 F:\OracleDB\目录下。最后结束 USERS 表空间的备份状态。

如果由于某种原因造成 USERS 表空间文件损坏或者丢失，此时启动数据库将会出现以下的错误。

```
SQL> STARTUP
ORACLE 例程已经启动。

Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              327156880   bytes
Database Buffers           96468992    bytes
Redo Buffers               6037504     bytes
数据库装载完毕。
ORA-01157: 无法标识/锁定数据文件 7 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 7: 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF'
```

解决办法，首先将备份好的数据备份文件复制到数据对应文件夹中进行覆盖，然后执行以下代码对数据进行恢复。

```
SQL> RECOVER DATAFILE 'F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF';
ORA-00279: 更改 2829795 (在 08/05/2011 11:28:15 生成) 对于线程 1 是必需的
ORA-00289: 建议:
F:\APP\ADMINISTRATOR\ORADATA\ORCL\ORCL00064 0640469817.001
ORA-00280: 更改 2829795 (用于线程 1) 在序列 #23 中
指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
auto
ORA-00279: 更改 2864572 (在 08/10/2011 12:02:50 生成) 对于线程 1 是必需的
ORA-00289: 建议:
F:\APP\ADMINISTRATOR\ORADATA\ORCL\ORCL00065 0640469817.001
ORA-00280: 更改 2864572 (用于线程 1) 在序列 #24 中
已应用的日志。
完成介质恢复。
```

在执行以上代码时，首先需要使用 STARTUP MOUNT 命令将数据库进行启动。



备份的数据文件是否可以修复？

网络课堂：http://bbs.itzcn.com/thread-662-1-1.html

在一次项目开发中，项目策划为该项目设定了一个数据库，具体的结构和框架已经成功的搭建，开始进行创建了。首先使用 CREATE TABLESPACE 语句在 E:\SHOPDB\目录下，代码如下。

```
SQL> CREATE TABLESPACE MYSHOP
  2  DATAFILE 'E:\SHOPDB\MYSHOP.DBF' SIZE 100M;
表空间已创建。
```

在该表空间中创建了很多个表，同时测试用的数据也插入到该表中，整整用了我一个多星期的时间，在创建完成时发生意外。数据库还没来得及备份时停电了，导致数据文件丢失，

是否有办法修复这些数据。还有就是原来创建数据库的磁盘也受到了损坏。

在这种情况下可以将数据进行恢复，首先第一步查询 V$RECOVER FILE，在该视图中存储了需要恢复的表空间信息，详细代码如下。

```
SQL> SELECT file#,error FROM v$recover file;

FILE#          ERROR
----------     ------------------------------
9              OFFLINE NORMAL
```

在代码查询结果中，FILE#表示错误表空间的表示值；ERROR 表示错误原因。

由于你创建的数据库没有进行备份，因此需要重新创建一个数据库在你原来的数据库位置上。可是你的磁盘被损坏了，需要将新创建的数据库创建在其他盘符中，最后通过使用 FILE#值对数据进行恢复。详细代码如下。

```
SQL> ALTER DATABASE CREATE DATAFILE 'E:\SHOPDB\MYSHOP.DBF' AS 'D:\
SHOPDB\MYSHOP.DBF';
数据库已更改。

SQL> RECOVER DATAFILE 9;
完成介质恢复。
```

到这里就已经成功地将数据库恢复了，然后将表空间状态设置为联机即可。

16.4.7 网络课堂



视频教学：http://school.itzcn.com/video-vid-1290-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16869-1-1.html

16.5 恢复控制文件出错

16.5.1 问题描述

大家都知道控制文件在数据库内起着很重要的作用，如果控制文件出现问题或者损坏，那么数据库就无法正常打开。当遇到控制文件损坏时，首先想到的就是对控制文件进行恢复。在 Oracle 数据库中有很多控制文件的镜像文件，只需要将镜像文件修改为控制文件名称即可。可是修复完成后，把数据库设置为 MOUNT 状态时出错，错误信息如下所示。

```
SQL> ALTER DATABASE MOUNT;
alter database mount
*
```

```
第 1 行出现错误:
ORA 00214: ???? '' F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL02.CTL'' ??
2022 ??? '' F:\APP\ADMINISTRATOR\ORADATA\ORCL\CONTROL01.CTL'' ?? 1902 ???
```

16.5.2 解决方法

这是因为两个控制文件之间的版本不同易引发的错误。解决方法只需要使用最新备份控制文件即可，具体恢复代码和步骤如下所示。

首先通过使用命令将新备份控制文件复制到数据库目录下，并且将其命名为和恢复的控制文件名称一致。

```
SQL> $COPY F:\APP\ADMINISTRATOR\ORADATA\ORCL\ CONTROL02.CTL
F:\APP\ADMINISTRATOR\ORADATA\ORCL\ CONTROL01.CTL;
已复制         1 个文件。
```

然后将数据库状态设置为 MOUNT，设置成功之后就表明数据库已经成功地修复，最后启动数据库即可完成此次恢复操作。代码如下。

```
SQL> ALTER DATABASE MOUNT;
数据库已更改。

SQL> ALTER DATABASE OPEN;
数据库已更改。
```

16.5.3 知识扩展——恢复控制文件

控制文件中记录了 Oracle 数据库的物理结构，也就是记录了数据库数据文件和日志文件的位置，控制文件中还记录了多种 SCN，用这些 SCN 来确定数据文件和日志文件是否是正确的，它的作用是指导数据库找到数据文件，日志文件并将数据库启动到打开状态。因此控制文件在 Oracle 数据库中起着很重要的作用，恢复控制文件也是必不可少的一项操作。

在操作 Oracle 数据库时，如果控制文件损坏或者丢失，那么首先采取恢复操作，在 Oracle 数据库中恢复控制文件有两种方法。

1．使用镜像副本恢复控制文件

在 Oracle 数据库中，为了确保控制文件的安全性，系统提供了多个副本控制文件，如果在操作数据库时，控制文件出现损坏或者其他原因无法正常使用控制文件。那么就可以通过副本控制文件对其进行恢复操作。

使用镜像文件恢复控制文件非常简单，只需要将镜像副本文件复制到控制文件目录下，将其名称更改为控制文件名称即可。



在复制副本恢复文件时，参数文件和控制文件的版本要同步，否则将会出现异常错误。

下面步骤就是使用镜像副本对控制文件进行恢复的操作：

（1）在进行恢复之前需要对控制文件进行备份，然后关闭数据库，代码如下。

```
SQL>$COPY F:\app\administrator\oradata\orcl\CONTROL01.CTL
  2  F:\app\administrator\oradata\orcl\CONTROL02.CTL
已复制         1 个文件
SQL>SHUTDOWN IMMEDIATE;
ORACLE 例程已经关闭。
```

（2）删除数据库下的 CONTROL01.CTL 控制文件，制作控制文件丢失现象。

（3）此时因为控制文件丢失，如果强行启动数据库将会显示错误信息，代码如下。

```
SQL> startup;
ORACLE 例程已经启动。

Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              348128400   bytes
Database Buffers           75497472    bytes
Redo Buffers               6037504     bytes
ORA-00205: error in identifying controlfile, check alertlog for more info
```

（4）将备份好的镜像文件 CONTROL02.CTL 通过命令复制到数据库目录下并且将名称修改为 CONTROL01.CTL，代码如下。

```
SQL>$COPY F:\app\administrator\oradata\orcl\CONTROL02.CTL
  2  F:\app\administrator\oradata\orcl\CONTROL01.CTL
已复制         1 个文件
```

（5）完成复制后，将数据库状态设置为 MOUNT，如果装载成功就说明已经成功地将控制文件进行了恢复。

```
SQL>ALTER DATABASE MOUNT;
数据库已更改。
```

（6）控制文件恢复成功之后可以通过使用以下代码打开数据库。

```
SQL>ALTER DATABASE OPEN;
数据库已更改。
```

2. 使用备份跟踪文件恢复控制文件

除了通过使用镜像副本文件恢复控制文件外，还可以通过跟踪文件进行恢复。所谓的跟踪文件就是一批 SQL 语句的组合。当控制文件损坏或者丢失时，通过这组 SQL 语句可以进行恢复操作，它的主要工作流程是通过 SQL 语句重新创建控制文件。



提示

在对控制文件进行恢复时，如果控制文件的副本文件也丢失，那么可以使用跟踪文件重新创建副本控制文件然后进行恢复。

在使用跟踪文件对控制文件进行恢复之前必须使用跟踪文件对控制文件进行备份，实现使用跟踪文件来对数据库控制文件进行恢复步骤如下所示。

（1）找到需要恢复的跟踪文件，将该文件另存为orcl ora 1976.sql，文件保存类型为ALL Files类型，然后打开文件将文件中所有说明进行注释，然后再进行保存。

（2）使用SHUTDOWN IMMEDIATE语句关闭数据库，然后将数据库中所有的镜像和备份控制文件进行删除。

（3）重新启动数据库，此时将会出现找不到控制文件而触发的异常信息，代码如下。

```
SQL> startup;
ORACLE 例程已经启动。

Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              348128400   bytes
Database Buffers           75497472    bytes
Redo Buffers               6037504     bytes
ORA-00205: error in identifying controlfile，check alertlog for more info
```

（4）使用START命令执行第1步中orcl_ora_1976.sql文件的语句代码，详细如下。

```
SQL> START D:\MyDB\orcl ora 1976.sql;
Oracle 历程已经启动。
Total System Global Area  431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              348128400   bytes
Database Buffers           75497472    bytes
Redo Buffers               6037504     bytes
控制文件已创建。
系统已创建。
数据库已创建。
表空间已创建。
```

（5）此时重新启动数据库，如果加载成功说明已经成功地将控制文件进行了恢复。

16.5.4 触类旁通



控制文件和镜像控制文件全部损坏怎么办？

网络课堂：http://bbs.itzcn.com/thread-16873-1-1.html

通常，如果控制文件损坏可以通过镜像文件对其进行恢复，可是由于磁盘问题导致镜像文件也损坏无法进行恢复操作，那么该如何解决。

如果在操作的过程中，镜像副本文件也损坏，那么还可以通过跟踪文件进行恢复。具体步骤如下所示。

（1）首先创建跟踪文件，语法如下所示。

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;

数据库已更改。
```

（2）查找新创建的跟踪文件位置，代码如下。

```
SQL> SHOW PARAMETER USER DUMP DEST;

NAME           TYPE       VALUE

user dump dest string     f:\app\administrator\diag\rdbms\orcl\orcl\trace
```

在查找结果的 VALUE 列值的位置查找最新修改文件，也就是新创建的跟踪文件。



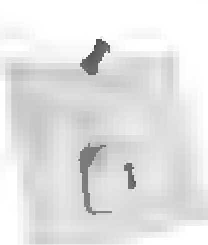
默认情况下，跟踪文件的默认名称包含数据库的名称、然后是一系列数字，从而保证文件名是唯一的，例如 orcl_ora_1728.trc。通常管理员都会重命名该文件，使其更容易识别。

（3）将新创建的跟踪文件后缀修改为 sql，类型修改为 All Files。打开该文件使用短线“--”注释所有说明，包括文件顶部的文本行，以及整个文档中在前面使用数字符号（#）或者（*）的所有行。但是不能注释创建文件的语句，然后再保存该文件。

（4）通过 START 命令重新创建控制文件，代码如下。

```
SQL> START F:\OracleDB\orcl ora 3996.sql
ORACLE 例程已经启动。
Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              327156880   bytes
Database Buffers           96468992    bytes
Redo Buffers               6037504     bytes
控制文件已创建。
系统已更改。
数据库已更改。
表空间已更改。
```

16.5.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1290-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16872-1-1.html

16.6 在操作数据库时，误删表如何恢复

16.6.1 问题描述

今天的项目要求设计数据库表，该表的主要作用是存储用户的其他信息，因此在命名上

和用户表非常相似，就因为这样所以才造成了误删表的事情。

在创建表时将名设置为userinfo_other用于存储用户的其他信息和用户信息表userinfo表名非常相似。当创建 userinfo_other 时创建错误，需要删除重新创建，可是不小心删除了userinfo 表。在该表中存储了大量的用户信息，是否有办法将其恢复。

16.6.2 解决方法

这样的问题解决办法通常就是恢复操作，如果你还记得删除表的时间，那么我建议使用基于时间的恢复操作。由于用户误操作对数据库做了错误的修改，或者删除了某个重要的表，如果用户知道操作的时间即可恢复到操作时间之前的位置。具体恢复步骤如下所示。

（1）在进行恢复操作之前，为了数据库的其他数据的安全，首先对数据库文件、日志文件和控制文件进行完全备份。

（2）然后关闭数据库并且断开所有用户的连接状态。然后从错误操作发生之前的最后一个备份中，将全部数据文件进行修复。然后启动数据库，将数据库设置为MOUNT状态。

（3）然后开始基于时间进行不完全的恢复操作，代码如下。

```
SQL> RECOVER DATABASE UNTIL TIME '2011-08-08 16:25:51';
完成介质恢复。
```

（4）然后打开数据库，代码如下。

```
SQL> ALTER DATABASE OPEN resetlogs;
数据库已更改。
```

（5）为了确保日志序列已经复位，可以通过以下语法进行查看。

```
SQL> ARCHIVE LOG LIST;
数据库日志模式          存档模式
自动存档                启用
存档终点                USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列      1
下一个存档日志序列      1
当前日志序列            1
```

（6）到这里就已经将误删的 userinfo 表成功地恢复了。但是还要注意的是，当执行RESETLOGS 之后，在开始备份的数据文件等备份内容就已经过期，需要重新做备份文件。

16.6.3 知识扩展——基于时间的恢复

基于时间的恢复数据库数据也是数据库恢复中比较方便的恢复操作。基于时间的恢复也称为时间点恢复。时间点恢复的主要流程是将用户在某一时间内的所有操作进行恢复。

在进行恢复时，语句如下所示。

```
RECOVER DATABASE UNTIL TIME time
```

以上语法中，time 表示需要恢复的具体时间，时间格式为“yyyy-mm-dd hh24:mi:ss”。

如果控制文件是通过备份修改的，那么在使用 RECOVER 语句时必须添加 USING BACKUP CONTROLFILE 选项，语法如下所示。

```
RECOVER DATABASE UNTIL TIME time USING BACKUP CONTROLFILE
```

基于时间的不完全恢复可以对表进行恢复也可以对表空间进行恢复。

1．对表的恢复

在对数据库表进行操作时，因某种原因导致数据库表数据丢失需要对该表进行恢复操作，那么通过基于时间对表的恢复时必须知道表数据丢失前的时间值。具体操作步骤如下所示。

（1）关闭当前数据库，将表数据丢失前最后一个备份数据库中的所有数据文件进行修复，最后将数据库设置为 MOUNT 状态，执行代码如下。

```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              348128400   bytes
Database Buffers           75497472    bytes
Redo Buffers               6037504     bytes
数据装载完毕。
```

（2）使用 RECOVER 命令对表进行基于时间的恢复操作，具体恢复到时间到 2011-09-21 08:25:00 时间段。

```
SQL> RECOVER DATABASE UNTIL TIME '2011-09-21 08:25:00';
完成介质恢复。
```

（3）介质恢复成功之后，别忘记了把数据库重新打开，代码如下。

```
SQL>ALTER DATABASE OPEN RESETLOGS;
数据库已更改。
```

执行到这里就已经成功地将表进行了恢复，上述代码中之所以使用 RESETLOGS，是因为执行完该语句后，数据库会重新建立重做日志，清空原有的重做日志文件内容，并且将日志序号修改为 1。



完成所有操作后，要备份数据文件和控制文件，因为在执行 RESETLOGS 之后，以前的备份就不能使用，所以需要重新对文件和控制文件进行备份。

2．对表空间的恢复

对表空间基于时间的恢复时需要使用备份的控制文件。在执行恢复时也必须知道该表空间被删除或者损坏的具体时间，这样恢复时恢复到该时间之前的时间段就可以保证数据的完

整性了。具体实现步骤如下。

（1）使用 SHUTDOWN IMMEDIATE 命令关闭数据库修复所有的数据文件和控制文件。

（2）使用 RECOVER 命令对表空间进行基于时间的恢复操作。因为发生损坏的时间为 2011-09-21 09:00:00，因此恢复时需要将时间恢复到该时间之前的时间段。实现代码如下。

```
SQL> RECOVER DATABASE UNTIL TIME '2011-09-21 08:50:00' USING BACKUP
CONTROLFILE;
……
```

由于执行提示信息过多，在这里省略运行提示信息，但是此次表空间恢复操作已经成功完成。

（3）最后是打开数据库，使用 RESELOGS 选项，代码如下。

```
SQL> ALTER DATABASE OPEN RESETLOGS;
数据库已更改。
```

16.6.4 知识扩展——基于更改的恢复

最常用最准确的恢复方式是基于更改的恢复了，基于更改的恢复是将数据库中提交的事务恢复到一个特定的系统修改序号（SCN）。而每个 SCN 号都对应了 Oracle 提交的数据，因此通过恢复最后一个 SCN 号可以完成基于修改的恢复。

在对数据库进行基于更改的恢复时，其使用语法如下。

```
RECOVER DATABASE UNTIL CHANGE scn;
```

上述语法中，scn 表示需要进行恢复的 SCN 号。具体基于更改恢复数据库步骤如下。

（1）使用 SELECT 语句来查询 V$LOGMNR_CONTENTS 视图内容来获取 SCN 号。

（2）使用 SHUTDOWN IMMEDIATE 命令关闭数。

（3）将数据库状态设置为 MOUNT 状态，代码如下。

```
SQL> STARTUP MOUNT;
数据库已更改。
```

（4）使用 RECOVER 语句对数据库进行基于更改的恢复，通过查询获取需要恢复的 SCN 号为 1075264，实现代码如下。

```
SQL> RECOVER DATABASE UNTIL CHANGE 1075264;
……
完成介质恢复。
```

（5）恢复完成后，使用 RESETLOGS 选项重新打开数据库，代码如下。

```
SQL>ALTER DATABASE OPEN RESETLOGS;
数据库已更改。
```

16.6.5 知识扩展——基于撤销的恢复

基于撤销的恢复就是将数据库恢复到特定的日志序列号之前的状态，如果联机日志文件由于某种原因导致损坏而又无法进行完全的数据库恢复时，可以进行基于撤销的不完全恢复，然后将数据恢复到撤销时对应的在以后一个重做日志文件后的状态。



执行基于撤销的不完全恢复时，首先要获取恢复的目标时刻，然后在看归挡重做日志文件的时间和日期标记。

当数据库文件损坏无法使用时，下面步骤中就是通过基于撤销的不完全恢复的方法，详细如下。

（1）使用 SHUTDOWN IMMEDIATE 语句关闭数据库，然后查看数据库日志文件，确定发生错误的归档时间。

```
Thu Jul 07 09:45:04 2011
Starting ORACLE instance (normal)
LICENSE MAX SESSION = 0
LICENSE SESSIONS WARNING = 0
Shared memory segment for instance monitoring created
Picked latch-free SCN scheme 2
……
```

从该文件可以得到删除操作的时间以及归档之日文件的位置和名称等信息。



数据库日志文件为 ALERT_ORCL.LOG，该文件存放在 Oracle 数据库安装目录的 diag\rdbms\orcl\orcl\trace 目录下。

（2）将数据库状态使用 STARTUP MOUNT 语句设置为 MOUNT 状态，对数据文件进行修复，代码如下。

```
SQL> startup mount;
ORACLE 例程已经启动。

Total System Global Area   431038464   bytes
Fixed Size                 1375088     bytes
Variable Size              348128400   bytes
Database Buffers           75497472    bytes
Redo Buffers               6037504     bytes
数据装载完毕。
```

（3）使用 RECOVER 语句执行基于撤销的不完全恢复操作，代码如下。

```
SQL> RECOVER DATABASE UNTIL CANCEL;
……
```

在该语句运行结果中，如果提示是否应用归档日志时，输入 ENTER 应用该归档日志即可。

如果找到包含错误的归档日志时，就可以输入 CANCEL 撤销对数据文件的应用，取消恢复过程。

16.6.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-1291-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16874-1-1.html

第 17 章　使用 RMAN 工具

在 Oracle 数据库中，数据的备份和恢复是比较重要的功能之一。为了方便开发人员对数据库进行备份操作，Oracle 还提供了一个 RMAN（Recovery Manager）恢复管理器工具。该工具是 Oracle 附带的工具，主要用于对数据进行备份、修复和恢复操作。

本章将会介绍如何使用 RMAN 工具对数据库数据进行备份、恢复的基本操作以及使用 RMAN 对数据库进行不同效果的恢复。

17.1　如何将 RMAN 资料档案保存在恢复目录中

17.1.1　问题描述

听说，Oracle 提供了一个 RMAN 工具用于对数据库的备份和恢复。但是，我怎么用不了这个工具啊？是不是还需要配置相关信息啊？

17.1.2　解决方法

是的，要使用 RMAN 工具，首先必须将配置的 RMAN 信息存储到恢复目录中。具体创建恢复目录步骤如下：

（1）使用数据库管理员登录 Oracle 数据，并为 RMAN 创建一个恢复目录数据库。

```
SQL> connect system/tiger
已连接。
SQL> CREATE TABLESPACE rman tablespace
  2  DATAFILE 'd:\db\rman tablespace.dbf' SIZE 100m;

表空间已创建。
```

（2）然后，为恢复目录创建数据库用户，并为该用户赋予相应的权限信息，代码如下：

```
SQL> CREATE USER test rman IDENTIFIED BY tiger
  2  DEFAULT TABLESPACE rman tablespace
  3  TEMPORARY TABLESPACE temp;
用户已创建。

SQL> GRANT CONNECT,RESOURCE TO test rman;
授权成功。
```

```
SQL> GRANT RECOVERY CATALOG OWNER TO test rman;
授权成功。
```

上述代码中，创建了一个 test rman 用户，设置用户默认表空间为 rman tablespace，然后为该用户赋予相应的操作权限。

（3）最后，使用新创建的 test rman 用户在恢复目录数据库中创建恢复目录。在【运行】窗口内输入 RMAN TARGET test rman/tiger 命令启动 RMAN 工具，详细代码如下所示。

```
连接到目标数据库: ORCL (DBID=1283643196)

RMAN> CONNECT CATALOG test rman/tiger
连接到恢复目录数据库

RMAN> CREATE CATALOG;
恢复目录已创建
```

17.1.3　知识扩展——RMAN 简介

RMAN 是 Oracle 数据库中自带的数据库备份恢复工具。当数据库需要进行备份时，可以通过使用该工具将数据库备份至磁盘上，在需要的情况下再使用该工具对其进行恢复。通过使用 RMAN 工具可以减少数据库管理员在对数据库进行备份产生的错误。

1．RMAN 特点

之所以在本书讲解使用 RMAN 工具备份数据库，是因为大多数的数据库管理员都会选择使用它来备份和恢复数据库。普通的数据库备份方式通过使用操作系统命令、SQL*Plus 命令、以及 SQL 语句来完成数据库备份与恢复。而 RMAN 工具只需要简单的 RMAN 命令就可以完成这份工作。下面讲解一下使用 RMAN 工具的优点。

❑ **跳过未使用的数据块**

使用 RMAN 的一个特点就是在备份数据库时，如果没有执行写入操作的数据块将不会进行备份。

❑ **备份压缩**

RMAN 使用一种 Oracle 特有的二进制压缩模式来节省备份设备上的空间。尽管传统的备份方法也可以使用操作系统的压缩技术，但 RMAN 使用的压缩算法是定制的，能够最大程度地压缩数据块中一些典型的数据。

❑ **执行增量备份**

如果不使用增量备份，那么每次 RMAN 都备份已使用块；如果使用增量备份，那么每次都备份上次备份以后发生变化的数据块，这样可以节省大量的磁盘空间、I/O 时间、CPU 时间和备份时间。

❑ **块级别的恢复**

RMAN 支持块级别的恢复，只需要还原或修复标识为损坏的少量数据块。在 RMAN 修复损坏的数据块时，表空间的其他部分以及表空间中的对象仍可以联机。

但是 RMAN 工具也有它自身不完美的地方，比如使用 RMAN 工具无法对非数据块进行

备份等。

2．RMAN 组件

RMAN 组件是和 Oracle 数据库客户端一起安装的客户端数据备份与恢复组件，通常 RMAN 命令在该组件中运行。简单的 RMAN 是由 RMAN 命令执行器和目标数据库组合而成，不过还有其他的一些组件，下面则是 RMAN 中常用的组件。

❑ **RMAN 命令执行器（RMAN Executable）**

RMAN 命令执行器和 Oracle 数据库中的 SQL*Plus 工具作用相同，RMAN 命令执行器用于接收 RMAN 命令从而完成数据的备份和恢复操作。

当打开一个 RMAN 程序时，系统将为 RMAN 创建一个用户进程，并在 Oracle 服务器上启动两个默认进程，分别用于提供与目标数据库的链接和监视远程调用。除此之外，根据会话期间执行的操作命令，系统还会启动其他进程。

启动 RMAN 命令执行器的详细过程如下。

（1）在 Windows 系统运行命令行中输入 RMAN 命令，如图 17-1 所示，单击【确定】按钮打开 RMAN 命令执行器窗口。



图 17-1 【运行】命令行窗口

（2）在打开的 RMAN.EXE 窗口中输入 SHOW ALL 命令将会打印出当前的 RMAN 的配置信息，如图 17-2 所示。



图 17-2 RMAN 信息窗口

此时，窗口中提示没有链接到目标数据库，那么用户可以在【运行】命令窗口中输入 RMAN TARGET system/tiger 命令指向需要链接的目标数据库，详细如图 17-3 所示。

打开【RMAN.EXE】窗口之后，在该窗口内再次输入 SHOW ALL 命令，此时在控制台中将会输出有关 RMAN 的配置信息。

```
Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

连接到目标数据库: ORCL (DBID=1283643196)

RMAN> SHOW ALL;

使用目标数据库控制文件替代恢复目录
db_unique_name 为 ORCL 的数据库的 RMAN 配置参数为:
CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default
CONFIGURE BACKUP OPTIMIZATION OFF; # default
CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default
CONFIGURE CONTROLFILE AUTOBACKUP OFF; # default
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '%F'; # default
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE MAXSETSIZE TO UNLIMITED; # default
CONFIGURE ENCRYPTION FOR DATABASE OFF; # default
CONFIGURE ENCRYPTION ALGORITHM 'AES128'; # default
CONFIGURE COMPRESSION ALGORITHM 'BASIC' AS OF RELEASE 'DEFAULT' OPTIMIZE FOR LOA
D TRUE ; # default
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE; # default
CONFIGURE SNAPSHOT CONTROLFILE NAME TO 'F:\APP\ADMINISTRATOR\PRODUCT\11.2.0\DBHO
ME_1\DATABASE\SNCFORCL.ORA'; # default
```

图 17-3　指定目标数据库 RMAN 信息窗口



可以在 RMAN 窗口中输入 EXIT 或 QUIT 命令，关闭（或退出）RMAN 执行器实用程序。

❑ **目标数据库（Target Database）**

目标数据库就是指将要被备份、恢复或者转存的数据库。在 RMAN 中，使用目标数据库控制文件来收集有关数据库的信息和 RMAN 的操作信息。

❑ **RMAN 恢复目录（RMAN Recover Catalog）**

RMAN 恢复目录是 RMAN 在 Oracle 数据库中创建的存储 RMAN 信息的对象。当使用 RMAN 对数据库进行备份或者恢复时，RMAN 会去读取控制文件中的数据库结构、归档日志等等备份信息，而这些信息没有存储到恢复目录内。

❑ **RMAN 恢复目录数据库（Recover Catalog Database）**

用来保存 RMAN 恢复目录的数据库，它是一个独立于目标数据库的 Oracle 数据库。

❑ **RMAN 资料档案库（RMAN Repository）**

在使用 RMAN 进行备份与恢复操作时，需要使用到的管理信息和数据称为 RMAN 资料档案库。资料档案库可以包括备份集、备份段、镜像副本、目标数据库结构和配置设置等内容。

17.1.4　知识扩展——RMAN 资料档案的保存

在 Oracle 数据库中，RMAN 的资料档案存储了备份和恢复的管理数据信息，因此它的存储也是非常重要的。而在 Oracle 数据库中它有两种存储方式，一种是保存在恢复目录中，另外一种则是保存在控制文件中。

1．保存在恢复目录中

恢复目录在前面也提到了，它是 RMAN 的一个可选组件。通常它被存放在一个独立的 Oracle 数据库中。当使用 RMAN 进行数据库备份或恢复时，它将会从恢复目录中读取数据信息。



将 RMAN 资料档案库保存在恢复目录中，具有一些优势，例如可以存储脚本，记录较长时间的备份恢复操作等。但是必须建立至少两个独立的数据库（目标数据库和恢复目录数据库）。如果无法满足这个基本条件，那么只能将 RMAN 资料档案库保存在目标数据库的控制文件中。

2．保存在控制文件中

如果 RMAN 资料档案库无法存储在恢复目录中，那么还可以将 RMAN 资料档案库存储在控制文件中。在把 RMAN 资料档案库存储在控制文件中时，目标数据库的控制文件中将包含两种类型的记录：

- **不可循环使用的记录**

用于存储记录一些比较重要而又不经常发生变化的数据信息。例如，数据库的结构信息。

- **可循环使用的记录**

用来记录非关键性的信息，存储的信息如果被存储满之后继续存储，将会覆盖之前的信息写入。



可循环使用的记录通常在数据库运行过程中不断生成，例如日志历史信息、归档重做日志文件、已建立的备份信息和脱机表空间的信息等，都是以循环使用的形式保存在控制文件中。

在控制文件中存储档案信息是有一定的限制的，如果存储的内容过多，空间存储完之后将会覆盖之前已经过期或者没用的记录信息。通常情况下，控制文件中的可循环使用记录至少需要保留 7 天才能被覆盖。

将 RMAN 控制文件配置为自动备份，可以避免当控制文件的所有副本都不可用时，RMAN 备份的信息丢失的现象。这样，无论何时使用备份命令或者数据库的结构发生变化，RMAN 都会备份控制文件。

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
使用目标数据库控制文件替代恢复目录
```

新的 RMAN 配置参数:

```
CONFIGURE CONTROLFILE AUTOBACKUP ON;
已成功存储新的 RMAN 配置参数
```

上述代码就是将 RMAN 启用自动备份，使用 CONFIGURE CONTROLFILE AUTOBACKUP ON 语句来完成。

17.1.5　知识扩展——配置 RMAN

在 RMAN 系统中，它自身有一套参数用于整个 RMAN 会话中，例如前面提到的 SHOW ALL 命令。通过该命令可以查看 RMAN 的一些系统参数的配置信息。数据库管理员可以使用 CONFIGURE 命令对 RMAN 进行配置，也可以在 CONFIGURE 命令中指定 CLEAR 关键字修改某选项的默认值等。

在配置 RMAN 时，通道的分配起着非常重要的作用，当服务器进程执行备份和恢复操作时，只有一个 RMAN 会话与分配的服务器恢复进行通信，每个通道的分配都会启动一个进程，

如图 17-4 是通道的使用图。

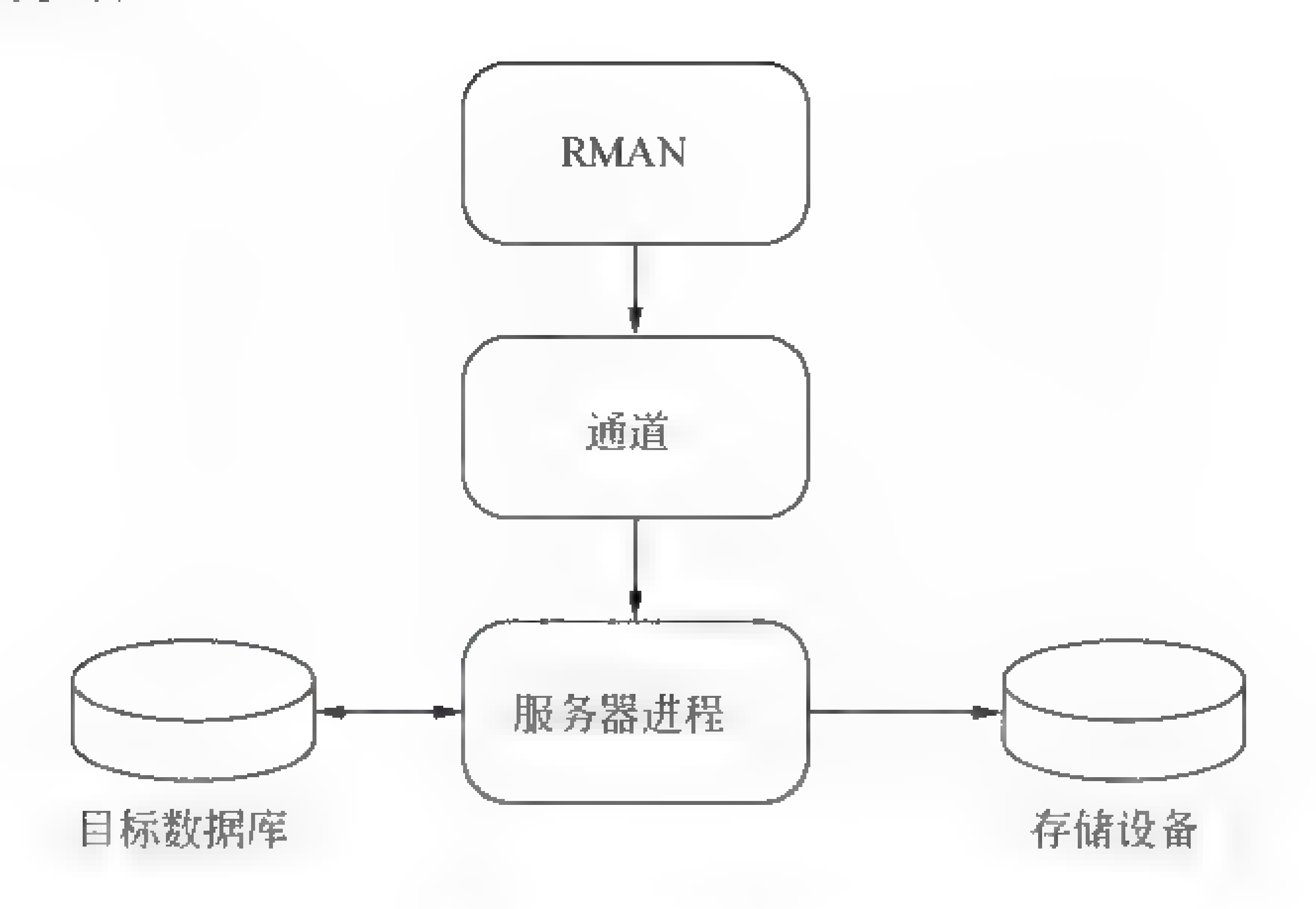


图 17-4　通道的使用

通道的分配分为自动分配通道和 RUN 命令手动分配通道，数据库管理员会根据不同的需求选择不同的通道分配方式。

1．手动分配通道

手动分配通道时必须使用 RUN 命令。RUN 命令的语法如下：

```
RUN {order;}
```

该语法中 order 表示命令。在 RMAN 中，RUN 命令会被优先执行，也就是说，如果 DBA 手动分配了通道，则 RMAN 将不再使用任何自动分配通道。例如下面实例代码就是对通道进行手动分配代码。

```
RMAN> RUN{
2> ALLOCATE CHANNEL run1 DEVICE TYPE DISK
3> FORMAT='D:\%u %c';
4> BACKUP TABLESPACE rman_tablespace CHANNEL run1
5> ;
6> }
分配的通道: run1
通道 run1: SID=146 设备类型=DISK
启动 backup 于 15-8 月 -11
通道 run1: 正在启动全部数据文件备份集
通道 run1: 正在指定备份集内的数据文件
输入数据文件: 文件号=00010 名称=D:\DB\RMAN_TABLESPACE.DBF
通道 run1: 正在启动段 1 于 15-8 月 -11
道 run1: 已完成段 1 于 15-8 月 -11
……
```


提示

当在 RMAN 命令执行器中执行类似 BACKUP、RESTOR 或 DELETE 等需要进行磁盘 I/O 操作时，可以将这些命令与 ALLOCATE CHANNEL 命令包含在一个 RUN 命令块内部。利用 ALLOCATE CHANNEL 命令为其手动分配通道。

2. 自动通道配置

在下面两种情况下，如果没有手动为 RMAN 分配通道，RMAN 将利用预定义的设置来自动分配通道：

- ❑ 在 RUN 命令块外部使用 BACKUP、RESTORE 和 DELETE 命令。
- ❑ 在 RUN 命令块内部执行 BACKUP 等命令之前，未使用 ALLOCATE CHANNEL 命令手动分配通道。

在使用自动分配通道时，RMAN 将根据下面这些命令的设置自动分配通道，如表 17-1 所示：

表 17-1　自动分配通道的预定义设置

[illegible]	[illegible]
CONFIGURE DEVICET TYPE sbt/disk PARALLELISM n;	用于设置自动通道个数
CONFIGURE DEFAULT DEVICE TYPE TO disk/sbt;	用于指定自动通道的默认设备
CONFIGURE CHANNEL DEVICE TYPE disk/sbt;	用于指定某一个通道的配置
CONFIGURE CHANNEL n DEVICE TYPE disk/sbt;	用于指定某一个通道的配置

例如下面实例中，使用 BACKUP 命令自动分配一个具有指定配置的通道，代码如下。

```
RMAN>BACKUP TABLESPACE users;
2>run {restore tablespace rman1;}
RMAN>CONFIGURE DEVICE TYPE disk PARALLELISM 3;
RMAN>CONFIGURE DEVICE TYPE sbt PARALLELISM 2;
```

在上述代码中就分配了一个自动通道代码，并且为 RMAN 分配了磁盘通道和磁带通道，disk 表示磁盘通道，而 sbt 就表示磁带通道。

3. 通道配置参数

通道配置参数是手动分配通道和自动分配通道都可以设置的功能，通过设置通道参数来控制通道备份时备份集的大小。

- ❑ **FILESPERSET 参数**　该参数用于限制执行 BACKUP 命令时备份集的文件个数。
- ❑ **CONNECT 参数**　用于设置数据库实例，RMAN 允许连接到多个不同的数据库实例。
- ❑ **FORMAT 参数**　该参数用于设置备份文件存储格式，以及备份文件的存储目录。表 17-2 列出了 FORMAT 格式化字符串以及各字符的意义。

表 17-2　FORMAT 格式化字符串

[illegible]	说　明
%c	表示备份段中的文件备份段号
%D	以 DD 格式显示日期
%Y	以 YYYY 格式显示年度
%n	在数据库名右边添加若干字母，构成 8 个字符长度的字符串。如 ora11g 自动形成为 ora11gXXX
%s	备份集号，此数字是控制文件中随备份集增加的一个计数器，从 1 开始
%T	指定年、月、日，格式为 YYYYMMDD
%U	指定一个便于使用的、由%u_%p_%c 构成的、确保不会重复的备份文件名称，RMAN 默认使用%U 格式
%d	指定数据库名

续表

[illegible]	[illegible]
%M	以 MM 格式显示月份
%F	结合数据库标识 DBID、日、月、年及序列，构成唯一的自动产生的字符串名字
%p	文件备份段号，在备份集中的备份文件片编码，从 1 开始每次增加 1
%t	指定备份集的时间戳，是一个 4 字节值的秒数值。%t 与%s 结合构成唯一的备份集名称
%u	指定备份集编码，以及备份集创建的时间构成的 8 个字符的文件名称
%%	指定字符串%，如%%Y 表示为%Y

- ❑ **RATE 参数**　用于设置通道的 I/O 限制。
- ❑ **MAXSETSIZE 参数**　用于配置备份集的最大尺寸。
- ❑ **MAXPIECESIZE 参数**　默认情况下一个备份集包含一个备份段，通过配置备份段的最大值，可以将一个备份集划分为几个备份段。
- ❑ **OPTIMIZATION 参数**　如果某个文件的完全相同的备份已经存在，那么当激活备份优化时，会跳过对该文件的备份。

17.1.6　触类旁通



如何实现手动分配通道？

网络课堂：http://bbs.itzcn.com/thread-16876-1-1.html

咨询一下，因为我是新接触 RMAN 工具，对通道的工作原理不怎么熟悉，今天到一家公司去面试，他让我使用 RMAN 工具实现一个手动通道的分配，我没有做出来。不知道大家是否知道如何分配，有知道的师哥师姐们帮忙实现个例子好吗？

通道的分配有手动和自动，如果选择手动分配通道需要使用 RUN 命令来完成，首先需要连接到目标数据库中，然后通过通道对表空间进行备份，详细实现代码如下。

```
RMAN> RUN{
2> ALLOCATE CHANNEL test run DEVICE TYPE DISK
3> FORMAT='D:\%u %c';
4> BACKUP TABLESPACE rman tablespace CHANNEL test run
5> ;
6> }

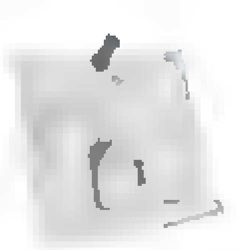
分配的通道: test_run
通道 test_run: SID=146 设备类型=DISK

启动 backup 于 15-8 月 -11
通道 test_run: 正在启动全部数据文件备份集
通道 test_run: 正在指定备份集内的数据文件
输入数据文件: 文件号=00010 名称=D:\DB\RMAN_TABLESPACE.DBF
通道 test_run: 正在启动段 1 于 15-8 月 -11
通道 test_run: 已完成段 1 于 15-8 月 -11
段句柄=D:\04MK2NGT_1 标记=TAG20110815T171053 注释=NONE
通道 test_run: 备份集已完成, 经过时间:00:00:01
完成 backup 于 15-8 月 -11
```

```
启动 Control File and SPFILE Autobackup 于 15-8月 -11
段 handle=F:\APP\ADMINISTRATOR\FLASH RECOVERY AREA\ORCL\AUTOBACKUP\
2011 08 15\O1
 MF S 759258654 74KRO04W .BKP comment=NONE
完成 Control File and SPFILE Autobackup 于 15-8月 -11
释放的通道: test_run
```

上述代码中，test_run 表示分配的通道名称，将其通道所创建的文件存储在 D 盘目录下，文件名称符合%u_%c 该规律，通过该通道将 rman_tablespace 表空间进行备份。

17.1.7 网络课堂



视频教学：http://school.itzcn.com/video-vid-1292-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1293-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-16875-1-1.html

17.2 链接 RMAN 时@的作用是什么

17.2.1 问题描述

我通常使用 RMAN 链接到目标数据库时，只需要在【运行】命令窗口输入 RMAN TARGET test_rman/tiger 命令即可。可是今天我在看一段视频教程时讲解老师链接 RMAN 到目标数据库时，在该命令后面又添加了@一串字符串。请问@表示什么意思，在这里有什么作用。

17.2.2 解决方法

如果链接到目标数据库时使用@符号，就说明链接的用户和目标数据库不在同一个数据库上。如果 RMAN 用户和目标数据库不在同一个数据库中时，则必须在链接的 TARGET 选项后使用@符号。例如以下代码的链接方式就是链接的用户名和目标数据库不同一个库下。

```
连接到目标数据库: ORCL (DBID=1283643196)
RMAN> CONNECT CATALOG system/tiger@orcl
连接到恢复目录数据库
```

17.2.3 知识扩展——RMAN 的基本操作

在 RMAN 里进行操作，几乎都需要通过命令才可以实现。不过，大家不需要担心，RMAN 的命令相对都比较简单，只要大家理解一下各个命令的含义就可以了。

1．RMAN 常用命令

在 RMAN 系统中，它的命令有很多，在前面也简单地为大家介绍了几个，下面在表 17-3

中列出了RMAN中最常用的几个命令。

表 17-3 RMAN 常用命令

RMAN 命令	说明
@	在@后指定的路径名处运行 RMAN 脚本。如果没有指定路径，则假定路径为调用 RMAN 所用的目录
STARTUP	启动目标数据库。相当于 SQL*Plus 中的 STARTUP 命令
RUN	运行"{"和"}"之间的一组 RMAN 语句，在执行该组语句时，允许重写默认的 RMAN 参数
SET	为 RMAN 会话过程设置配置信息
SHOW	显示所有的或单个的 RMAN 配置
SHUTDOWN	从 RMAN 关闭目标数据库。相当于 SQL*Plus 中的 SHUTDOWN 命令
SQL	运行那些使用标准的 RMAN 命令不能直接或间接完成的 SQL 命令
ADVISE FAILURE	显示针对所发现故障的修复选项
BACKUP	执行带有或不带有归档重做日志的 RMAN 备份。备份数据文件、数据文件副本或执行增量 0 级或 1 级备份。备份整个数据库或一个单独的表空间或数据文件。使用 VALIDATE 子句来验证要备份的数据库
CATALOG	将有关文件副本和用户管理备份的信息添加到存储库
CHANGE	改变 RMAN 存储库中的备份状态。可以用于显式地从还原或恢复操作中排除备份，或者将操作系统命令删除了备份文件的操作通知 RMAN
CONFIGURE	为 RMAN 配置持久化参数。在接下来的每个 RMAN 会话中这些配置参数都是有效的，除非显式地清除或修改它们
CONVERT	为跨平台传送表空间或整个数据库而转换数据文件个数
CREATE CATALOG	为一个或多个目标数据库创建包含 RMAN 元数据的存储库目录。强烈建议不要将该目录存储在其中的一个目标数据库中
CROSSCHECK	对照磁盘或磁带上的实际文件，检查 RMAN 存储库中的备份记录。将对象标识为 EXPIRED、AVAILABLE、UNAVAILABLE 或 OBSOLETE。如果对象对 RMAN 是不可用的，那么把它标识为 UNAVAILABLE
DELETE	删除备份文件或副本，并在目标数据库控制文件中将它们标识为 DELETED。如果使用了存储库，将清除备份文件的记录
DROP DATABASE	从磁盘删除目标数据库，并反注册数据库
DUPLICATE	使用目标数据库的备份来创建副本数据库
FLASHBACK	执行 FLASHBACK DATABASE（闪回数据库）操作
LIST	显示在目标数据库控制文件或存储库中记录的有关备份集和映像副本的信息
RECOVER	对数据文件、表空间或者整个数据库执行完全的或不完全的恢复。还可以将增量备份应用到一个数据文件映射副本，以便在时间上向前回滚该副本
REGISTER DATABASE	在 RMAN 存储库中注册目标数据库
REPAIR FAILURE	修复自动诊断存储库中记录的一个或多个故障
REPORT	对 RMAN 存储库进行详尽的分析
RESTORE	通常在存储介质失效后，将文件从映像副本或备份集恢复到磁盘上
TRANSPORT TABLESPACE	为一个或多个表空间的备份创建可移植的表空间集
VALIDATE	检查备份集并报告它的数据是否原样未动，以及是否一致

2．链接目标数据库

在 RMAN 中链接数据库，实际上就是在 RMAN 系统和 Oracle 数据库之间建立关系。通过这种关系来操作数据库。同时链接的方式也有两种：无恢复目录链接方式、有恢复目录链接方式。

❑ **无恢复目录**

在无恢复目录下链接数据库可以使用 RMAN TARGET 语句、RMAN NOCATALOG 语句和 RMAN TARGET … NOCATALOG 语句来完成。例如以下代码，在【运行】命令窗口中输入 CMD，然后使用其中任意语句来完成链接，代码如下所示。

```
C:\Documents and Settings\Administrator>cd\

C:\>RMAN TARGET system/tiger NOCATALOG
恢复管理器: Release 11.2.0.1.0 - Production on 星期二 8月 16 09:15:51 2011
Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.
连接到目标数据库: ORCL (DBID=1283643196)
使用目标数据库控制文件替代恢复目录

RMAN>
```

在上述代码中，使用了 RMAN TARGET … NOCATALOG 语句来完成无恢复目录的链接方式。首先需要进入 C 盘根目录下，然后输入命令。在使用该语句链接数据库时还要注意的是被链接的数据库服务必须为启动状态，否则将会抛出异常信息。

❑ **有恢复目录**

有目录的链接方式在前面也简单的介绍过，首先使用 RMAN TARGET 命令打开 RMAN 工具；然后使用 CONNECT CATALOG 语句建立与恢复目录的连接；接着使用 REGISTER DATABASE 命令注册目标数据库；最后使用 RESYNC CATALOG 命令来实现恢复目录与目标数据库控制文件之间的同步。

3．取消数据库注册

如果需要解除被注册的数据库，那么可以通过使用两种不同的方法来取消数据库注册。

❑ 使用 UNREGISTER 命令。

❑ 使用 DBMS_RCVCAL.UNREGISTERDATABASE()过程。

其中，使用过程取消数据库注册需要使用数据库 DB_KEY 与 DB_ID 值，用户可以查询 DB 表获取 DB_KEY 与 DB_ID 信息。语法如下所示。

```
EXEC DBMS_RCVCAL.UNREGISTERDATABASE(db_key,db_id)
```

例如下面实例代码中，首先需要查询 DB 表中的 DB_KEY 与 DB_ID 数据信息，然后通过该信息进行数据库的注销操作。代码如下。

```
SQL>SELECT DB_KEY,DB_ID FROM DB;
DB_KEY              DB_ID
------------------- -------------------
1                   2525452535
```

```
SQL> EXEC DBMS RCVCAT.UNREGISTERDATABASE(1,2525452535);
PL/SQL 过程已成功完成。
```

4. 在 RMAN 中执行 SQL 语句

用户在操作 RMAN 工具时，可能有时需要同时执行一个 SQL 语句，那么就需要打开 Sql Plus 工具，在该工具中执行 SQL 语句。RMAN 为了开发者更方便地操作数据库，在 RMAN 工具中也可以执行 SQL 语句，执行时必须使用 SQL 命令来完成。

例如下面代码中，使用 SQL 命令在 RMAN 工具中执行删除表和删除数据等操作，代码如下。

```
RMAN> SQL 'DROP TABLE teacher PURGE';
sql 语句: drop table teacher purge
RMAN> SQL "DELETE FROM users WHERE username=''小明''";
sql 语句: delete from users where username=''小明''
```

5. 备份压缩

数据库备份是经常被使用到的操作，如果在执行数据库备份时，发现备份文件过大，控制不够使用，怎么办。别急大家可以对备份文件进行压缩。备份压缩需要使用 COMPRESSED 命令来完成。例如以下代码。

```
RMAN>CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COMPRESSED BACKUPSET;
……
```

17.2.4 触类旁通



使用过程取消数据库注册具体步骤是什么？

网络课堂：http://bbs.itzcn.com/thread-16878-1-1.html

新手上路，望各位大哥大姐相互照应下。遇到一个棘手的问题，不知道怎么去做，希望大家能够帮我解决一下。问题要求使用过程来注销数据库，具体步骤该如何去做，各位大哥帮我做下好吗？小弟我在线等。

使用 DBMS_RCVCAL.UNREGISTERDATABASE()过程取消数据库注册具体步骤如下所示。

（1）使用恢复目录用户链接数据库，代码如下所示。

```
SQL> CONNECT test rman/tiger
已连接。
```

（2）查询数据库的相关 DB_KEY 和 DB_ID 的信息，代码如下所示。

```
SQL> SELECT DB KEY,DB ID FROM DB;
DB KEY      DB ID
----------  ----------------------
1           1580625356
……
```

（3）调用 DBMS RCVCAL.UNREGISTERDATABASE()过程传入参数，执行取消注册操作。

```
SQL> EXECUTE DBMS RCVCAT.UNREGISTERDATABASE(1,1580625356);
PL/SQL 过程已成功完成。
```

17.2.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1294-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1295-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1297-spid-35.html
网络课堂：http://bbs.itzcn.com/thread-16877-1-1.html

17.3 增量备份的好处以及如何执行增量备份

17.3.1 问题描述

对于一个大文件，增量备份时发现它和上次备份的记录不同，我是否可以仅仅把这个文件增加或者修改过的地方追加到上次备份的文件中？那么增量备份是否可以做到这点，具体如何实现该操作的？

17.3.2 解决方法

当然可以实现。增量备份就是将发生改变的数据块添加到备份集中进行追加备份。通过使用增量备份可以满足你所需要的备份方式，具体实现增量备份步骤如下。

（1）使用用户对象在 RMAN 中链接需要进行备份的目标数据库，代码如下所示。

```
RMAN> CONNECT CATALOG system/tiger@orcl

连接到恢复目录数据库
```

（2）对数据库进行注册操作，使用 REGISTER DATABASE 命令，运行结果代码如下所示。

```
RMAN> REGISTER DATABASE;

注册在恢复目录中的数据库
正在启动全部恢复目录的 resync
完成全部 resync
```

上述代码是注册了数据库。用户也可以选择对表空间或者单独的数据库文件进行备份。如果需要查看已经备份好的数据库信息，可以使用以下代码进行查询。

```
RMAN> REPORT NEED BACKUP DAYS=2;

文件报表的恢复需要超过 2 天的归档日志
文件天数据   名称
---- ----- -----------------------------------------------------
1    502   F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSTEM01.DBF
2    502   F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSAUX01.DBF
3    502   F:\APP\ADMINISTRATOR\ORADATA\ORCL\UNDOTBS01.DBF
4    502   F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF
5    41    F:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF
6    41    F:\APP\ADMINISTRATOR\ORADATA\ORCL\TEST.DBF
7    41    F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF
8    36    F:\APP\ADMINISTRATOR\ORADATA\ORCL\TESTUSERS.DBF
9    12    D:\SHOPDB\MYSHOP.DBF
```

（3）为了确保数据的安全，可以执行一次完全的增量备份，具体实现代码如下所示。

```
RMAN> BACKUP INCREMENTAL LEVEL 0
2> AS COMPRESSED BACKUPSET DATABASE;

启动 backup 于 17-8 月 -11
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: SID=133 设备类型=DISK
通道 ORA_DISK_1: 正在启动压缩的增量级别 0 数据文件备份集
通道 ORA_DISK_1: 正在指定备份集内的数据文件
输入数据文件: 文件号=00001 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSTEM01.DBF
输入数据文件: 文件号=00002 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSAUX01.DBF
输入数据文件: 文件号=00006 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\TEST.DBF
输入数据文件: 文件号=00005 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF
输入数据文件: 文件号=00007 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS.DBF
输入数据文件: 文件号=00009 名称=D:\SHOPDB\MYSHOP.DBF
输入数据文件: 文件号=00010 名称=D:\DB\RMAN TABLESPACE.DBF
输入数据文件: 文件号=00003 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\UNDOTBS01.DBF
输入数据文件: 文件号=00004 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF
输入数据文件: 文件号=00008 名称=F:\APP\ADMINISTRATOR\ORADATA\ORCL\TESTUSERS.DBF
通道 ORA_DISK_1: 正在启动段 1 于 17-8 月 -11
通道 ORA_DISK_1: 已完成段 1 于 17-8 月 -11
段句柄 F:\APP\ADMINISTRATOR\FLASH RECOVERY AREA\ORCL\BACKUPSET\2011 08 17\O1 MF
NNND0_TAG20110817T101255_74P8XBVL_.BKP 标记=TAG20110817T101255 注释=NONE
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:01:45
完成 backup 于 17-8 月 -11

启动 Control File and SPFILE Autobackup 于 17-8 月 -11
段 handle=F:\APP\ADMINISTRATOR\FLASH_RECOVERY_AREA\ORCL\AUTOBACKUP\2011_08_17\O1
 MF S 759406484 74P90OTL .BKP comment NONE
完成 Control File and SPFILE Autobackup 于 17-8 月 -11
```

在这段代码中，设置 INCREMENTAL 选项默认为差异增量备份。如果需要设置累积增量备份只需要在命令中设置 CUMULATIVE 选项即可。

17.3.3 知识扩展——RMAN 备份类型

RMAN 是一个专业的数据库备份工具，在 RMAN 中对数据库进行备份的类型也有很多种。例如下面是两种比较常用的备份类型：完全备份（Full Backup）和增量备份（Incremental Backup）等。

1．完全备份

顾名思义，完全备份是将除空白数据块外的所有数据块、控制文件和日志文件全部进行备份。执行完完全备份后，还可以执行其他备份操作。

2．增量备份

在进行增量数据备份时，除空白数据块外 RMAN 会将发生改变的数据块进行备份操作，而没有任何变化的数据块则不进行任何操作。增量备份的范围可以是单独的数据文件、表空间或者全部数据库。

其中增量备份的方式又分为两种：

- ❑ **差异备份** 差异备份是默认的备份方式，在备份时需要使用 DIFFERENTIAL 关键字，它是将备份上一次进行的同级或者低级备份以来所有变化的数据块。
- ❑ **累积备份** 使用累积备份时需要使用 CUMULATIVE 关键字，它将备份上次低级备份以来所有的数据块。



采用累积备份还是差异备份，在一定程度上取决于 CPU 周期，以及磁盘的可用空间。使用累积备份意味着备份文件将会变得日益庞大，并花费更长的时间，但是在一次还原与恢复过程中，只需要两个备份集。使用差异备份只记录从上次备份以来的变化，但是如果从多个备份集进行恢复，这种操作可能会花费更长的时间。

3．增量备份的方式

增量备份通过两种方式来实现，如表 17-4 所示。

表 17-4 增量备份的两种方式

[illegible]	[illegible]	[illegible]	[illegible]
差异备份	DIFFERENTIAL	是	将备份上一次进行的同级或者低级备份以来所有变化的数据块
累积备份	CUMULATIVE	否	将备份上次低级备份以来所有的数据块

例如，在一周之内每天使用的增量备份级别为：0 级、2 级、2 级、2 级、1 级、2 级和 2 级，下面分别使用这两种备份方式，实现不同的备份效果。

（1）使用差异增量备份的方式，备份效果如图 17-5 所示。

周日进行一次 0 级增量备份，RMAN 将数据文件中所有非空白的数据块都复制到备份集中。

周一进行级别 2 的差异方式增量备份，由于不存在任何最近一次建立的级别为 2 或级别为 1 的增量备份，RMAN 会对周日建立的 0 级增量备份相比较，将发生变化的数据块保存到备份集中，即备份周日以后发生变化的数据。

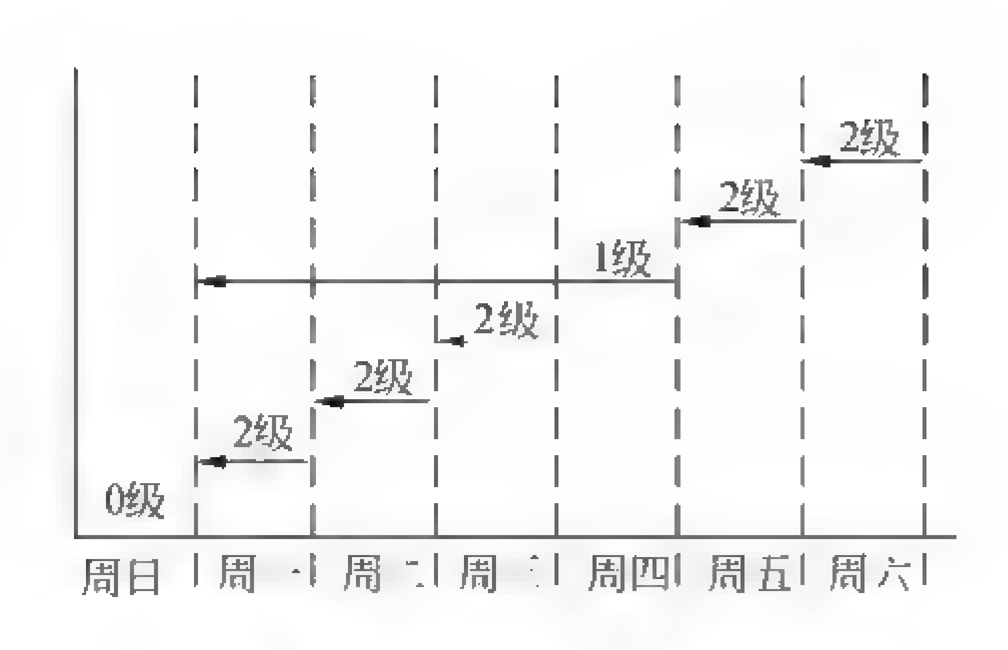


图 17-5　差异增量备份

周二进行级别为 2 的差异增量备份，将备份周一以后发生变化的数据。

周四进行级别为 1 的差异增量备份，RMAN 将与周日建立的级别为 0 的增量备份相比，将那些发生变化的数据块保存到备份集中。

周五进行一次级别为 2 的差异增量备份，RMAN 只备份从周四开始发生变化的数据。

周六进行一次级别为 2 的差异增量备份，RMAN 只备份从周五开始发生变化的数据。



使用上述差异增量备份的好处是：如果周五发生故障，则只需要利用周四的 1 级备份和周日的 0 级备份，即可完成对数据库的恢复。

（2）如果使用累积增量备份的方式，备份效果如图 17-6 所示。

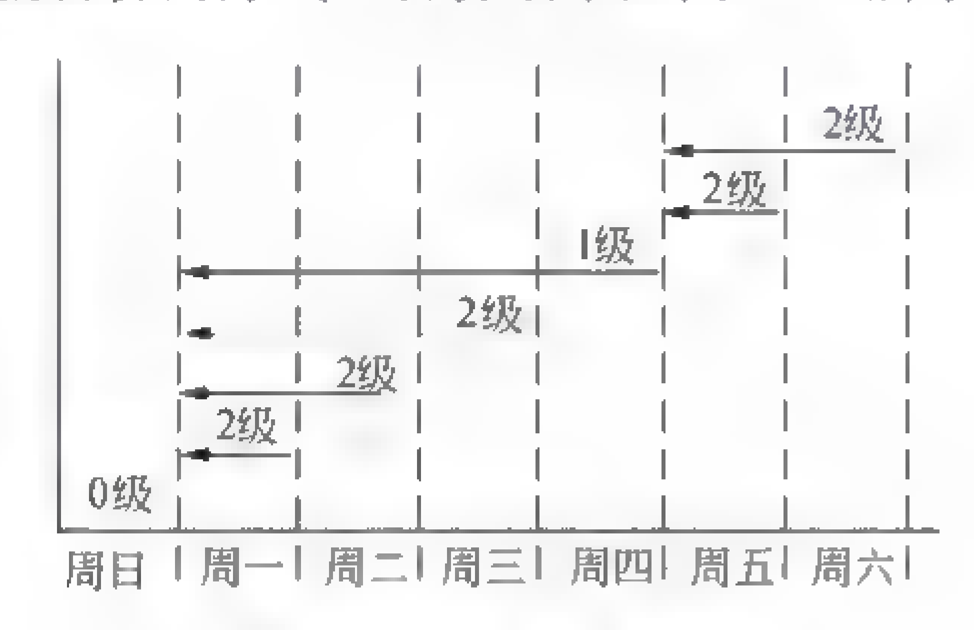


图 17-6　累积增量备份

周日进行一次级别为 0 的累积增量备份，RMAN 将数据文件中所有非空白数据块保存在备份集中。

周一进行级别为 2 的累积增量备份。由于不存在任何最近一次建立的 1 级增量备份，RMAN 以周日的 0 级增量备份作为基准，将发生变化的数据块保存到备份集中。即只备份从周日以来发生变化的数据。

周二进行级别为 2 的累积增量备份，RMAN 将备份从周日开始发生变化的数据。



在周二建立的 2 级增量备份中，实际上包含了周一的 2 级增量备份，因此这种增量备份方式称为累积方式。

周四进行级别为 1 的累积增量备份。RMAN 将以周日建立的 0 级增量备份为基准，将之后发生变化的数据块保存到备份集中。

19.2.2 解决方法

如果你所使用的 Oracle 版本足够高的话，可以使用闪回表技术来将表中的数据恢复到你修改之前的时间点数据。假设你修改之前的时间为 2011-08-19 12:13:14，则可以使用下面的语句将其恢复：

```
FLASHBACK TABLE table name TO timestamp
to_timestamp('2011-08-19 12:13:14','YYYY-MM-DD HH24:MI:SS');
```

19.2.3 知识扩展——使用闪回表

如果某个用户不小心删除了一个十分重要的表，后果将十分严重，在 Oracle 9i 中提供的闪回特性只能回复 DML 语句造成的影响，而无法回复 DDL 语句造成的影响。DBA 只能通过重建一个表，然后从备份数据中导入。利用 Oracle 10g 中的闪回表技术，DBA 可以轻松地恢复表的数据。

闪回表实质上是将表中的数据恢复到指定的时间点（TIMESTAMP）或系统改变号（SCN）上，并将自动恢复索引、触发器和约束等属性，同时数据库保持联机，从而增加整体的可用性。

1. 撤销表空间

闪回表需要用到数据库中的撤销表空间，可以通过 SHOW PARAMETER undo 语句查看与 UNDO 表空间相关的信息。如下：

```
SQL> SHOW PARAMETER undo;
NAME                    TYPE          VALUE
--------------------  -----------  ------------------------------
undo management          string        AUTO
undo retention           integer       900
undo_tablespace          string        UNDOTBS1
```

其中，undo_management 表示系统的撤销数据管理方式，其值为 AUTO 则表示系统使用自动撤销管理方式，也就是使用撤销表空间记录撤销数据，其值为 MANUAL 则表示系统使用回退段撤销管理方式；undo_retention 表示撤销数据在撤销表空间中的保留时间；undo_tablespace 表示所使用的撤销表空间的名称。



用户对表数据的操作都记录在撤销表空间中，这为表的闪回提供了数据恢复的基础。例如，某个操作在提交后被记录在撤销表空间中，保留时间为 900 秒，用户可以在这 900 秒内对表进行闪回操作，从而将表中的数据恢复到操作之前的状态。

因为撤销表空间的空间大小是有限的，所以需要设置记录的保留时间，这样可以保证记录的操作是最新发生的。如果用户创建的撤销表空间足够大，则可以 ALTER SYSTEM 语句将保留时间设置长一些，如下：

```
...
6           白雪                22              女
已选择 6 行。
```



注意

FLASHBACK TABLE 不可以用于备份数据库上，也不可以闪回对表的 DDL 操作。如果想要将表闪回到某个系统改变号上，可以使用 TIMESTAMP TO SCN()函数将时间戳转换为 SCN。但是 SCN 不容易把握，用户很难知道应该闪回到哪个 SCN 上，相对来说，时间戳就明显很多。

19.2.4 触类旁通



恢复表中数据时报 ORA-08189 的错误，如何解决？

网络课堂：http://bbs.itzcn.com/thread-16768-1-1.html

我需要将 EMP 表中的数据恢复到 2011-08-19 14::44:00 时间点，于是执行了下面的语句：

```
SQL>FLASHBACK TABLE emp TO timestamp
 2  to_timestamp('2011-08-19 14:44:00','yyyy-mm-dd hh24:mi:ss');
```

结果报出如下的错误：

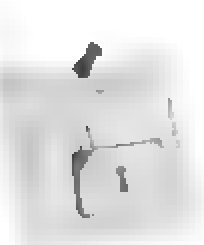
```
ORA-08189: 因为未启用行移动功能，不能闪回表
```

从报错提示我明白需要启动行移动功能？闪回表之前必须要启动行移动功能吗？怎么启动？

在使用 FLASHBACK TABLE ...语句之前必须启动行移动功能，语句如下：

```
ALTER TABLE EMP ENABLE ROW MOVEMENT;
```

19.2.5 网络课堂



视频教学：http://school.itzcn.com/video-vid-1311-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16767-1-1.html

19.3 如何恢复被误删除的表及表数据

19.3.1 问题描述

我不小心删除了一个表，并且我也没有定期做备份，如何才能将我删除的表及数据恢复到我的数据库中？这个表的数据是很重要的，我考虑过定期备份的方法，但是备份的周期太长，不太乐观。在 Oracle 中还有什么方法可以用于恢复数据库表？

19.3.2 解决方法

通过一个例子来说明这个问题的解决方法吧！首先我使用 DROP TABLE 语句将一个表删除：

```
DROP TABLE test;
```

这时候再用 SELECT 语句查询此表时，将会提示表或视图不存在。但可以用如下查询语句查询到这个表在 Oracle 回收站中：

```
SELECT * FROM user_recyclebin WHERE original_name='test';
```

如果是这样，我们可以使用如下的语句进行恢复：

```
FLASHBACK TABLE test TO BEFORE DROP;
```

19.3.3 知识扩展——使用闪回删除

实现闪回删除功能，需要使用 Oracle 回收站（RecycleBin）。回收站是所有被删除对象及其相依对象的逻辑存储容器，例如当一个表被删除（DROP）时，该表及其相依对象（包括：索引、约束、触发器、嵌套表、LOB 和 LOB 索引段等）并不会马上被数据库彻底删除，而是被保存到回收站中。

被删除的对象名称可能是相同的，为确保添加到回收站中的对象名称的唯一性，系统会对这些保存到回收站中的对象进行重命名，命名格式如下：

```
BIN$globalUID$version
```

其中，globalUID 是一个全局唯一的、24 个字符长的标识对象，该标识与原对象名没有任何关系；$version 指数据库分配的版本号。

闪回删除可以将回收站中被删除的对象进行还原，语法格式如下：

```
FLASHBACK TABLE table name
TO BEFORE DROP [ RENAME TO new_table_name ] ;
```

其中，table_name 可以使用表的原名，也可以使用表在回收站中的名称。如果表的原名相同，则在使用原名进行闪回删除操作时，默认还原最近一次删除的表。表被还原后，默认情况下使用其原名，而如果该名称已经存在，则需要在还原该表时对其重命名，这时需要使用 RENAME TO 子句。

下面通过一个例子来具体介绍如何使用闪回删除技术将回收站中保存的对象进行还原。

（1）在 SCOTT 模式下创建一个名称为 mytable 的表，并向表中添加两条记录。如下：

```
SQL> CREATE TABLE mytable(
  2  test VARCHAR2(50));
表已创建。
```

```
SQL>  INSERT INTO mytable
  2  VALUES(
  3   '使用闪回删除技术将回收站中的对象还原');
已创建 1 行。
SQL> INSERT INTO mytable
  2  VALUES(
  3  '使用闪回删除技术将回收站中的对象还原');
已创建 1 行。
```

（2）先以 DBA 的身份连接数据库，并使用 PURGE 命令对回收站进行清空。然后再使用 SCOTT 用户的身份连接数据库，并使用 DROP TABLE 语句将 mytable 表删除，如下：

```
SQL> purge dba recyclebin;
DBA 回收站已清空。
SQL> DROP TABLE mytable;
表已删除。
```

（3）通过 USER_RECYCLEBIN 数据字典视图查看回收站中的信息，如下：

```
SQL> SELECT original name , object name , type , droptime
  2  FROM user recyclebin;

ORIGINAL NAME    OBJECT NAME                    TYPE         DROPTIME
--------------- ----------------------------- ------- ---------------------
MYTABLE        BIN$VGPMvqv+TrC4JLi1beqP+g==$0  TABLE  2011-08-20:14:42:27
```

（4）使用 FLASHBACK 语句还原该表，为了防止表名重复，这里使用 RENAME 对其进行重命名为 mytable2，如下：

```
SQL> FLASHBACK TABLE  mytable
  2  TO BEFORE DROP
  3  RENAME TO mytable2;

闪回完成。
```

（5）使用 SELECT 语句查询 mytable2 表，检查是否成功还原，如下：

```
SQL> SELECT * FROM mytable2;
TEST
--------------------------------------------------
使用闪回删除技术将回收站中的对象还原
使用闪回删除技术将回收站中的对象还原
```

19.3.4　知识扩展——管理回收站

闪回删除技术是将回收站中保存的对象进行还原，它依赖于回收站。那么本节将对回收站的管理进行全面的讲解。

```
SQL> DESC test;
 名称                                        是否为空?           类型
 ----------------------------------------- -------------- --------------------------
 NAME                                                          VARCHAR2(20)
SQL> SELECT object name,original name,droptime FROM recyclebin;
OBJECT_NAME                          ORIGINAL_NAME      DROPTIME

BIN$m/QBxED5SY2bpGftJtK9oA==$0    TEST                 2011-08-24:10:11:49
```

从结果可以看出，FLASHBACK 语句还原的是最后放入回收站中的表。

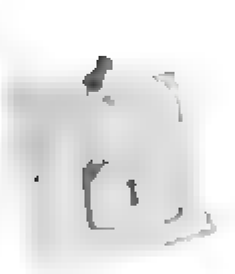
（3）再次删除还原之后的 test 表，请清除回收站中的表，如下：

```
SQL> DROP TABLE test;
表已删除。
SQL>  SELECT object name,original name,droptime FROM recyclebin;
OBJECT NAME                          ORIGINAL NAME      DROPTIME
------------------------------ --------------- -------------------
BIN$G0ldHTfNTG+CLZmvevK33A==$0   TEST             2011-08-24:10:13:43
BIN$m/QBxED5SY2bpGftJtK9oA==$0   TEST             2011-08-24:10:11:49
SQL> PURGE TABLE test;
表已清除。
SQL>  SELECT object_name,original_name,droptime FROM recyclebin;
OBJECT NAME                          ORIGINAL NAME         DROPTIME
------------------------------ ---------------- -------------------------
BIN$G0ldHTfNTG+CLZmvevK33A==$0   TEST              2011-08-24:10:13:43
```

从结果来看，PURGE 语句清除的是最早进入回收站中的表。

当回收站中存在同名对象的时候，可以用回收站中的名字进行清除或还原。另外，FLASHBACK 还有 RENAME TO 语句，可以在还原的时候对表进行重命名，避免和当前用户下已经存在的表冲突。

19.3.6 网络课堂



视频教学：http://school.itzcn.com/video-vid-1312-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1313-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1314-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1315-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16769-1-1.html

19.4 如何确定对数据表的操作时间

19.4.1 问题描述

在 Oracle 表闪回时，我们经常要根据时间点和 SCN 号。我怎么确定要闪回的时间或 SCN

号呢？比如我在6:08:12的时候更新了一条表中的数据，然后在6:08:30的时候删除了一条记录。那么我要把表闪回到我更新记录以前，我该如何做？更新和删除操作仅仅差十几秒，我怎么样把握这么精确的时间呢？如果我不想根据时间闪回，而想根据SCN闪回，那么我怎样确定或查询到更新和删除操作时的SCN呢？

19.4.2 解决方法

Oracle的时间与SCN有一个对应关系，这个对应关系可利用函数timestamp_to_scn()和scn_to_timestamp()实现。在Oracle 11g中，时间与SCN的对应精确度已经可以精确到3秒了。对于Oracle的查询技术，可以使用闪回版本查询（FLASHBACK VERSION QUERY）。如表EMP，在2011年8月19日15:00更新一条数据，其EMPNO为7782，而在同一天的14:00点又将该条数据删除了，则可以使用查询：

```
SELECT versions_operation , versions_starttime , versions_endtime ,empno ,
ename, sal , deptno
FROM emp
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE empno = 7782;
```

查询结果为：

```
V... OPERATION  V... STARTTIME  V... ENDTIME    EMPNO ENAME  SAL      DEPTNO
D               2011-8-19 14:00                 7782  SMITH  1,000.00  20
U               2011-8-19 15:00 2011-8-19 15:13 7782  SMITH  1,000.00  20
```

从查询结果可以获取到更新表和删除表的时间，另外可以使用VERSIONS_STARTSCN字段和VERSIONS_ENDSCN字段获取到更新表和删除表的时间所对应的SCN。

19.4.3 知识扩展——使用闪回版本查询

闪回版本查询技术用于查询某段时间内对表的操作记录，主要针对INSERT、UPDATE和DELETE操作。闪回版本查询的语法形式如下：

```
SELECT column [ , … ] FROM table name
VERSIONS
{
    BETWEEN SCN | TIMESTAMP expr | MINVALUE AND expr | MAXVALUE
    |
    AS OF SCN | TIMESTAMP expr
} ;
```

语法说明如下：

- ❑ **column_name** 列名。
- ❑ **table_name** 表名。

- **BETWEEN ... AND** 时间段，或系统改变号段。
- **SCN** 系统改变号。
- **TIMESTAMP** 时间戳。
- **MAXVALUE** 最大值。
- **MINVALUE** 最小值。
- **expr** 指定一个值或表达式，表示某个时间点或系统改变号。
- **AS OF** 表示恢复单个版本。

下面以一个示例的形式，介绍如何使用闪回版本查询获取对表的操作记录。

（1）向 SCOTT 模式下的 DEPT 表中添加两条记录，并将 DEPNO 为 40 的 DNAME 值修改为 IT，如下：

```
SQL>  SELECT * FROM dept;
DEPTNO         DNAME            LOC
---------- -------------- ----------------
10             ACCOUNTING       NEW YORK
20             RESEARCH          DALLAS
30             SALES             CHICAGO
40             OPERATIONS        BOSTON
SQL>  INSERT INTO dept
  2   VALUES(50,'ORACLE','CHINA');
已创建 1 行。
SQL> COMMIT;
提交完成。
SQL> INSERT INTO dept
  2  VALUES(60,'JAVA','NEW YORK');
已创建 1 行。
SQL> COMMIT;
提交完成。
SQL>  UPDATE dept SET dname='OPERATIONS' WHERE deptno=40;
已更新 1 行。
SQL> COMMIT;
提交完成。
```

如上述示例，每执行一次 DML 操作后都使用 COMMIT 命令进行提交。

再次查询 DEPT 表中的内容，如下：

```
SQL> SELECT * FROM dept;
DEPTNO         DNAME          LOC

10             ACCOUNTING     NEW YORK
20             RESEARCH        DALLAS
30             SALES           CHICAGO
40             OPERATIONS      BOSTON
50             ORACLE         CHINA
60             JAVA            NEW YORK
```

19.5.2 解决方法

这个问题可能是由于 Oracle 11g 默认把 SUPPLEMENTAL LOGGING 禁用了导致的。使用如下语句，可以把 SUPPLEMENTAL LOGGING 打开就好了：

```
ALTER DATABASE ADD supplemental log DATA;
```

19.5.3 知识扩展——使用闪回事务查询

闪回事务查询提供了一种查看事务级数据库变化的方法。实现闪回事务查询，需要先了解 FLASHBACK_TRANSACTION_QUERY 视图，从该视图中可以获取回滚段中存储的事务信息。FLASHBACK_TRANSACTION_QUERY 视图的结构如下：

```
SQL> DESC flashback transaction query;
 名称                             是否为空?        类型
 --------------------------- --------- -----------------------------
 XID                                              RAW(8)
 START SCN                                        NUMBER
 START TIMESTAMP                                  DATE
 COMMIT SCN                                       NUMBER
 COMMIT TIMESTAMP                                 DATE
 LOGON USER                                       VARCHAR2(30)
 UNDO CHANGE#                                     NUMBER
 OPERATION                                        VARCHAR2(32)
 TABLE_NAME                                       VARCHAR2(256)
 TABLE_OWNER                                      VARCHAR2(32)
 ROW ID                                           VARCHAR2(19)
 UNDO_SQL                                         VARCHAR2(4000)
```

字段说明如下：

- **XID** 事务标识。
- **START_SCN** 事务起始时的系统改变号。
- **START_TIMESTAMP** 事务起始时的时间戳。
- **COMMIT_SCN** 事务提交时的系统改变号。
- **COMMIT_TIMESTAMP** 事务提交时的时间戳。
- **LOGON_USER** 当前登录用户名。
- **UNDO_CHANGE#** 撤销改变号。
- **OPERATION** 前滚操作，也就是该事务所对应的操作。
- **TABLE_NAME** 表名。
- **TABLE_OWNER** 表的拥有者。
- **ROW_ID** 唯一的行标识。
- **UNDO_SQL** 用于撤销的 SQL 语句。



使用闪回事务查询需要用户具有 SELECT ANY TRANSACTION 权限。

下面举例说明闪回事务查询的使用。

（1）向 SCOTT 模式下的 STUDENT 表中添加一条记录，如下：

```
SQL> INSERT INTO student
  2  VALUES(7,'袁娜',22,'女');
已创建 1 行。
SQL> COMMIT;
提交完成。
```

（2）修改 STUDENT 表中 ID 为 1 的 NAME 列值为“马向林”，如下：

```
SQL> UPDATE student SET name='马向林'
  2  WHERE id=1;
已更新 1 行。
SQL> COMMIT;
提交完成。
```

（3）下面使用闪回版本获取事务 ID，这需要用到另外一个伪列：VERSIONS_XID，如下：

```
SQL> SELECT id,name,versions_operation,versions_xid
  2  FROM student
  3  versions BETWEEN timestamp minvalue AND maxvalue;

ID          NAME        VERSIONS OPERATION    VERSIONS XID
---------- ------------ --------------------- -----------------
1          马向林          U                    0100090080040000
1          张瑞
...
6          白雪
7          袁娜             I                    01000E0082040000
已选择 8 行。
```

其中，VERSIONS_XID 字段表示事务的标识，通过它可以唯一指定某个事务。

（4）使用闪回事务查询获取“01000E0082040000”所对应的事务信息，如下：

```
SQL>  SELECT table name,operation,undo sql
  2   FROM flashback transaction query
  3    WHERE xid='01000E0082040000';
TABLE NAME              OPERATION              UNDO SQL

STUDENT                  INSERT                delete from "SCOTT"."STUDENT"
                                                where ROWID = 'AAASOtAAEAAAAJFAAB';
```

- ❑ **TO TIMESTAMP** 指定一个时间戳。
- ❑ **expr** 指定一个值或表达式。
- ❑ **TO BERORE SCN** 前滚到指定的 SCN。
- ❑ **TO BEFORE TIMESTAMP** 前滚到指定的时间戳。

为了实现数据库的前滚，Oracle 提供了一组闪回日志（Flashback Logs），它记录了数据库的前滚操作。使用 SHOW PARAMETER DB_RECOVERTY_FILE_DEST 命令可以查看闪回日志信息（要求用户为管理员身份），如下：

```
SQL>  SHOW PARAMETER db_recovery_file_dest;
NAME                         TYPE          VALUE
---------------------------- ------------ ---------------------------
db_recovery_file_dest        string        D:\app\Administrator\flash_
                                           recovery_area
db_recovery_file_dest_size   big integer   3852M
```

其中：DB_RECOVERY_FILE_DEST 表示闪回日志的存放位置；DB_RECOVERY_FILE_DEST_SIZE 表示存放闪回日志的空间（即恢复区）的大小。

下面通过一个示例来具体介绍如何使用闪回数据库，示例如下：

（1）以管理员的身份连接数据库，并通过数据字典 V$DATABASE 查看当前闪回数据库功能是否已经启用，如下：

```
SQL> SELECT flashback_on FROM v$database;
FLASHBACK_ON
------------------
NO
```

数据字典 V$DATABASE 的 FLASHBACK_ON 字段表示闪回数据库功能是否启用，如果该字段值为 YES，则表示启用；如果为 NO，则表示未启用。

（2）查看当前数据库的日志模式是否为归档模式，如下：

```
SQL> ARCHIVE LOG LIST;
数据库日志模式         非存档模式
自动存档               禁用
存档终点               USE_DB_RECOVERY_FILE_DEST
最早的联机日志序列       50
当前日志序列             52
```

关闭数据库，启动为 MOUNT 状态，使用 ALTER DATABASE ARCHIVELOG 语句将数据库由非归档模式修改为归档模式，并打开数据库。再次使用 ARCHIVELOG LIST 查看数据库的日志模式如下：

```
SQL> ARCHIVE LOG LIST;
数据库日志模式         存档模式
自动存档               启用
存档终点               USE_DB_RECOVERY_FILE_DEST
```

```
ORACLE 例程已经启动。

Total System Global Area  431038464 bytes
Fixed Size                  1375088 bytes
Variable Size             327156880 bytes
Database Buffers           96468992 bytes
Redo Buffers                6037504 bytes
数据库装载完毕。
SQL> FLASHBACK DATABASE TO
  2  timestamp(TO CHAR('2011-08-22 18:44:56','YYYY-MM-DD HH24:MI:SS'));
闪回完成。
```

（6）使用 ALTER SYSTEM OPEN RESETLOGS 语句打开数据库，如下：

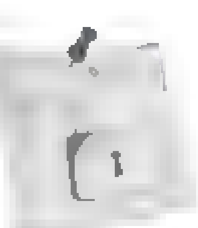
```
SQL> ALTER DATABASE OPEN RESETLOGS;
数据库已更改。
```

（7）查询 SCOTT.EMP 表中的数据是否已经恢复，如下：

```
SQL> SELECT * FROM SCOTT.EMP;
EMPNO      ENAME     JOB        MGR    HIREDATE     SAL    COMM      DEPTNO
-------- -------- --------- ---- ----------- ----- -------- -------
7369       SMITH     CLERK     7902   17-12 月-80   800     20
7499       ALLEN     SALESMAN 7698   20-2 月-81    1600    300        30
...
7934      MILLER    CLERK     7782   23-1 月 -82  1300    10
已选择 14 行。
```

从查询结果可以看出，EMP 表中的数据已经恢复，这说明数据库已经成功闪回到了指定的时间点上。

19.6.4 网络课堂



视频教学：http://school.itzcn.com/video-vid-1318-spid-35.html
视频教学：http://school.itzcn.com/video-vid-1319-spid-35.html

网络课堂：http://bbs.itzcn.com/thread-16773-1-1.html

19.7 系统报 ORA-55612 错误

19.7.1 问题描述

在创建闪回数据归档区时，系统报 ORA-55612 的错误，如下：

```
SQL> CREATE FLASHBACK ARCHIVE myarch
```

```
  2  TABLESPACE users
  3  QUOTA 20M
  4  RETENTION 10 month;
TABLESPACE users
          *
第 2 行出现错误:
ORA-55612: 无权管理闪回归档
```

我使用的是 SCOTT 用户创建的，该用户没有创建闪回数据归档区的权限吗？创建闪回数据归档区需要哪些权限？

19.7.2　解决方法

在 Oracle 11g 中引入了一个新的闪回技术——闪回数据归档，同时也引入了一个新的权限——FLASHBACK ARCHIVE ADMINISTER。创建与修改闪回数据归档区需要用户具有 FLASHBACK ARCHIVE ADMINISTER 系统权限。

19.7.3　知识扩展——创建与管理闪回数据归档区

闪回数据归档区，是指存储闪回数据归档的历史数据的区域，它是一个逻辑概念，其实质是从一个或多个表空间中分出一定的空间来保存对象的数据修改操作。闪回数据归档并不是针对数据库的所有变化，而是可以根据需要指定某些数据库对象，将这些指定对象的变化数据保存在闪回数据归档区中，这就极大地减少了存储空间的大小。

1. 创建闪回数据归档区

一个 Oracle 数据库中可以有多个闪回数据归档区，但最多只允许存在一个默认闪回数据归档区，各个闪回数据归档区都可以有自己的数据管理策略，例如都可以设置自己的数据保留时间等，互不影响。

虽然闪回数据归档区可以基于多个表空间，但是在创建时只能为其指定一个表空间，如果需要指定多个，可以在创建之后使用 ALTER 语句进行添加。创建与修改闪回数据归档区需要用户具有 FLASHBACK ARCHIVE ADMINISTER 系统权限。

创建闪回数据归档区的语法形式如下：

```
CREATE FLASHBACK ARCHIVE [ DEFAULT ] archive_name
TABLESPACE tablespace_name [ QUOTA size K | M ]
RETENTION retention_time ;
```

语法说明如下：

- ❑ **DEFAULT**　指定创建默认的闪回数据归档区。要求用户具有 SYSDBA 系统权限。
- ❑ **archive_name**　闪回数据归档区的名称。
- ❑ **TABLESPACE**　为闪回数据归档区指定表空间。
- ❑ **QUOTA**　为闪回数据归档区分配最大的磁盘限额。如果不使用此选项，则闪回数据归档区的磁盘限额将受表空间中的磁盘限额限制。



```
SQL> ALTER FLASHBACK ARCHIVE archive 01
  2  MODIFY RETENTION 3 day;
闪回档案已变更。
```

❑ 删除闪回数据归档区

例如删除闪回数据归档区 archive_01，如下：

```
SQL> DROP FLASHBACK ARCHIVE archive 01;
闪回档案已删除。
```

19.7.4 知识扩展——为表指定闪回数据归档

闪回数据归档区可以针对一个或多个数据库对象，在为一个数据库对象指定归档区时一般有两种情况：一种是在创建对象时为其指定归档区，另一种是为已存在的对象指定归档区。


注意

为表指定闪回数据归档区后，将不允许对该表执行 DDL 操作，例如删除表、增加或删除列、重命名等。

1．创建表时指定闪回数据归档区

例如，现有一个名称为 archive_02 的闪回数据归档区，我们需要在创建 table1 时为其指定闪回数据归档区为 archive_02，如下：

```
SQL> CREATE TABLE table1(
  2  id NUMBER,
  3  content VARCHAR2(50))
  4  FLASHBACK ARCHIVE archive 02;
表已创建。
```

2．为现有表指定闪回数据归档区

例如现有一个名称为 table2 的表，我们为其指定闪回数据归档区为 archive_02，如下：

```
SQL> ALTER TABLE table2
  2  FLASHBACK ARCHIVE archive 02;
表已更改。
```


注意

在使用 FLASHBACK ARCHIVE 子句为表指定闪回数据归档区时，如果不明确指定闪回数据归档区的名称，则表示使用默认闪回数据归档区，而如果数据库中没有默认闪回数据归档区，则 Oracle 返回错误。

3．取消表的闪回数据归档区

为表指定闪回数据归档区后，对表的操作将受到限制，例如不允许删除表等。使用 ALTER TABLE 语句可以取消表的闪回数据归档区，其语法形式如下：

```
ALTER TABLE table_name NO FLASHBACK ARCHIVE ;
```

例如取消 table2 表的闪回数据归档区 archive_02，如下：

```
2011 08 23 10:59:36
```

可以看出，test01 表和 test02 表中的数据是在 10:59:36 写进去的。按照数据保存策略来看，表 test01 没有数据归档区，而表 test02 数据应当保存在自定义的数据归档区 archive 02，保存期限为 5 天。

（4）使用 DELETE 命令对表进行删除操作，并查询删除之后表中的数据，如下：

```
SQL> DELETE FROM test01;
已删除 3 行。
SQL> DELETE FROM test02;
已删除 3 行。
SQL> COMMIT;
提交完成。

SQL> SELECT * FROM test01;
未选定行
SQL> SELECT * FROM test02;
未选定行
```

（5）利用 FLASHBACK 功能去查询两个表在 2011-08-23 10:59:36 时间点上的内容，如下：

```
SQL> SELECT * FROM test01
  2  AS OF timestamp
  3  to timestamp('2011-08-23 10:59:36','YYYY-MM-DD HH24:MI:SS');
NUM
----------
1
2
3
SQL> SELECT * FROM test02
  2  AS OF timestamp
  3  to timestamp('2011-08-23 10:59:36','YYYY-MM-DD HH24:MI:SS');
NUM
----------
4
5
6
```

从查询结果可以获得 2011-08-23 10:59:36 时间点上的内容，但是不能确认的是，这些数据是 UNDO 撤销数据提供的，还是闪回数据归档区提供的。

（6）为了证明获取的数据是闪回数据归档区提供的，我们来创建一个新的撤销表空间，并将系统使用的撤销表空间切换为新的撤销表空间，同时将原来的撤销表空间 UNDOTBS1 删除，如下：

```
SQL> CREATE UNDO TABLESPACE undo02
  2  DATAFILE 'D:\app\Administrator\oradata\orcl\undotbs02.dbf'
```

第 20 章　银行卡电子交易系统

银行卡对于大多数人都不再那么陌生了，生活中银行卡的身影无处不在。上街购物、吃饭或者做生意都会使用到银行卡。之所以这么多用户都选择使用银行卡的原因是携带方便，如果需要消费时只需要数据密码即可，不需要用户带现金购物。

那么很多人都非常了解银行卡的操作和使用，但是并不了解在使用或者操作银行卡时后台做的一些工作，以及钱是如何扣除和存入的。本节为大家介绍使用 Oracle 数据库实现银行卡消费系统。

20.1 系统分析

本章所要实现的是一个银行卡消费系统，通过该系统结合本节所讲解的知识点进行案例实现。经过对系统的分析大致可分为以下功能：银行卡用户管理、银行卡交易管理和交易记录管理等综合功能。

20.1.1 需求分析

根据需求为该系统指定了以下 9 项功能：用户开户、用户激活、删除用户、修改用户信息、存款业务、取款业务、转账业务、交易记录查询和交易记录删除等功能，这 9 项功能的功能简介如下。

- **用户开户**　向数据库用户表中添加一条字段，表示一个用户信息。
- **用户激活**　默认新开户用户处于等待激活状态，通过该功能进行银行卡激活。
- **删除用户**　在用户使用过程中，如果需要注销该用户则可以执行该功能。
- **修改用户**　用户在使用时，可以通过该功能对其信息进行修改。
- **存款业务**　将用户现金存储到银行卡中。
- **取款业务**　将银行卡中钱提取现金。
- **转账业务**　将用户卡中金额减少，转向用户金额增加。
- **交易记录查询**　用户做出任何一项交易都会有相应的记录信息。
- **交易记录删除**　如果有无效的交易记录可以进行删除操作。

通过上面对系统的分析后系统模块结构如图 20-1 所示。

20.1.2 系统设计

通过对该系统的结构和功能的分析，为该系统定制了 3 个系统表分别为：用户信息表（users）、银行卡表（card）、业务类型表（types）、交易记录表（record）和黑名单表（blacklist）。

```
  2  DATAFILE'D:\mydb\banks.dbf' SIZE 500M
  3  AUTOEXTEND ON NEXT 100M MAXSIZE 1000M
  4  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M
  5  SEGMENT SPACE MANAGEMENT MANUAL;

表空间已创建。
```

在上述代码中，使用 SYSTEM 用户登录，在 D:\mydb\目录下创建了名为 banks 的数据库，并且设置该数据库为 500M，最大可用空间为 1000M 等参数配置。创建完成之后会在 D:\mydb\目录下生成一个大于 500M 的 banks.dbf 文件，如图 20-2 所示。

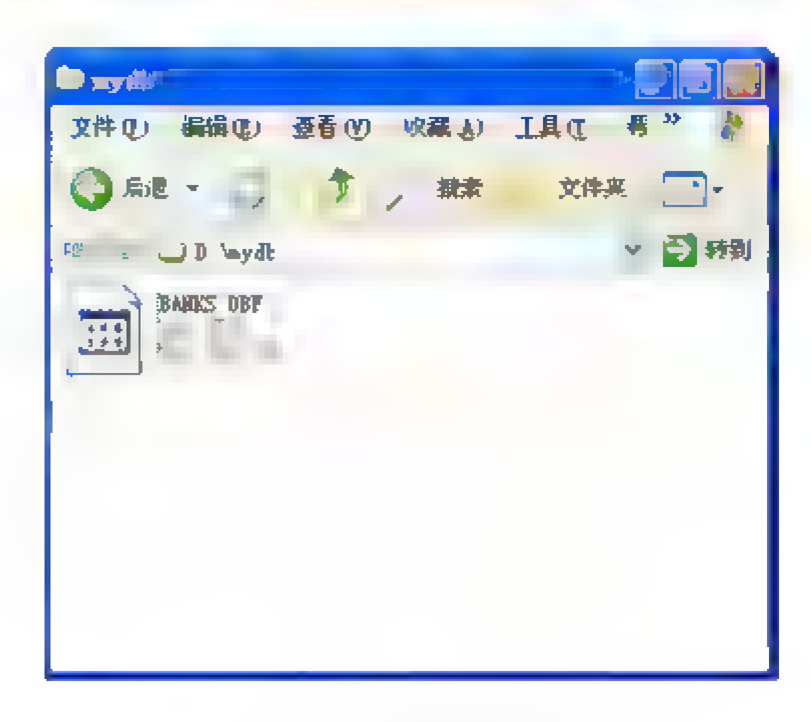


图 20-2　BANKS 数据库

数据库创建成功之后为该数据创建一个特殊用户，该用户主要用于管理该数据库信息。例如以下代码创建了名为 bankadmin 的用户，并且设置密码为 000000。详细代码如下所示。

```
SQL> CREATE USER bankadmin
  2  IDENTIFIED BY 000000
  3  DEFAULT TABLESPACE banks
  4  TEMPORARY TABLESPACE temp
  5  QUOTA 30M ON banks
  6  PASSWORD EXPIRE;
用户已创建。

SQL> GRANT CONNECT,RESOURCE TO bankadmin;
授权成功。

SQL> ALTER USER bankadmin ACCOUNT UNLOCK;
用户已更改。
```

上述代码中，首先创建了 bankadmin 用户，然后给用户设置密码，用户创建成功之后给用户赋予登录权限，并且解锁该用户。

20.2.2　创建表

用户和相应的表空间创建完成之后，就开始创建数据库的结构部分，那就是数据库表。

数据库表在一个 Oracle 系统中起着非常重要的作用，如果没有数据库表的存在，那么数据库的存在也是没有任何实质的作用。下面对前面所设计的 4 个表进行创建。

1．users 表

用户表几乎每个系统都会有 users 表的身影的出现，该表用于存储用户的基本信息。其中 ID 为 NUMBER 类型自增列，添加数据时用户不需要为该列插入数据，系统将会自动生成数据值。详细创建代码如下。

```
SQL> CREATE TABLE users(
  2  id NUMBER NOT NULL,
  3  username VARCHAR2(10) NOT NULL,
  4  usersex VARCHAR2(4) NOT NULL,
  5  userage NUMBER NOT NULL,
  6  userphone VARCHAR2(20) NOT NULL,
  7  useraddress VARCHAR2(50) NOT NULL,
  8  CONSTRAINT u pk PRIMARY KEY(id)
  9  )TABLESPACE banks;

表已创建。
```

以上代码创建了 users 表，但是该表的 ID 暂时还不会自动生成，需要为该列设置触发器，在插入数据时触发该触发器向该列中插入一个数据值。在创建触发器之前需要创建序列，序列中存储了从 1～10000 的数据值，代码如下。

```
SQL> CREATE SEQUENCE "se users"
  2  MINVALUE 1 MAXVALUE 10000
  3  INCREMENT BY 1
  4  START WITH 1
  5  CACHE 20
  6  NOORDER CYCLE;

序列已创建。
```

序列创建成功之后，则开始创建触发器。当数据库表插入数据时将会触发该触发器从序列中读取值插入到 ID 列中，代码如下。

```
SQL> CREATE OR REPLACE TRIGGER tr users
  2  BEFORE INSERT ON users
  3  FOR EACH ROW
  4  WHEN (NEW.ID IS NULL)
  5  BEGIN
  6  SELECT se users.NEXTVAL INTO :NEW.ID FROM DUAL;
  7  END;
  8  /
触发器已创建
```

触发器创建成功之后用户可以向表中插入数据，ID 列会自动生成数据值，到这里该表已

经成功地创建，在图 20-3 中列出了 users 表的结构图。



图 20-3 users 表结构图

2. card 表

card 表中存储了用户银行卡信息，创建表是为 ID 列设置为主键唯一，同时表中的 cardMONEY 列的数据不可小于 0，USERID 和 users 表中的 ID 列有着主外键关系。详细创建代码如下。

```
SQL> CREATE TABLE card(
  2  id NUMBER NOT NULL,
  3  cardnumber VARCHAR2(20) NOT NULL,
  4  cardpwd VARCHAR2(6) NOT NULL,
  5  cardmoney NUMBER NOT NULL CONSTRAINT record ck CHECK(cardmoney>=0),
  6  cardtypeid NUMBER NOT NULL,
  7  userid NUMBER NOT NULL REFERENCES users(id),
  8  CONSTRAINT c pk PRIMARY KEY(id)
  9  )TABLESPACE banks;

表已创建。
```

表创建完成之后同样需要为该表创建序列和触发器，用于生成主键 ID 列的数据值，在这里创建一个名为 se_card 序列和名为 tr_card 的触发器，详细代码如下。

```
SQL> CREATE SEQUENCE "se card"
  2  MINVALUE 1 MAXVALUE 10000
  3  INCREMENT BY 1
  4  START WITH 1
  5  CACHE 20
  6  NOORDER CYCLE;
序列已创建。

SQL> CREATE OR REPLACE TRIGGER tr card
  2  BEFORE INSERT ON card
  3  FOR EACH ROW
  4  WHEN (NEW.ID IS NULL)
  5  BEGIN
```

```
    5  BEGIN
    6  SELECT se type.NEXTVAL INTO :NEW.ID FROM DUAL;
    7  END;
    8  /
触发器已创建
```

代码中，创建了序列和触发器，当用户向该表插入数据时将会生成 ID 值。图 20-5 中列出了该表的结构信息。



图 20-5　types 表结构图

表创建完成之后需要向该表中插入测试数据，在这里向表中分别插入取款业务、存款业务、转账业务和消费业务，代码如下。

```
INSERT INTO types VALUES('取款业务');
已创建 1 行。
INSERT INTO types VALUES('存款业务');
已创建 1 行。
INSERT INTO types VALUES('转账业务');
已创建 1 行。
INSERT INTO types VALUES('消费业务');
已创建 1 行。
```

用户可以通过使用 SELECT 语句查看数据是否已经成功地插入到了该表中，运行结果如图 20-6 所示。



图 20-6　types 表数据

4．record 表

record 表用于存储用户消费记录信息，当用户进行不同业务操作时，会在该表中存储一套数据信息，该表 5 列其中有 2 列为外键，具体创建代码如下。

```
SQL> CREATE TABLE record(
```

5. blacklist 表

该表中存储了违反银行卡使用条款的用户信息，表中有用户 ID 列为被添加的用户 ID 信息以及添加时间和添加原因等，详细创建代码如下。

```
SQL> CREATE TABLE blacklist(
  2  id NUMBER NOT NULL,
  3  userid NUMBER REFERENCES users(id),
  4  reason VARCHAR2(100),
  5  btime DATE NOT NULL,
  6  CONSTRAINT b pk PRIMARY KEY(id)
  7  )TABLESPACE banks;

表已创建。
```

代码中创建了 blacklist 表，并且为该表中 userid 列设置为外键，并且将 id 列设置为主键，接下来创建序列和触发器来实现主键自增效果。代码如下。

```
SQL> CREATE SEQUENCE "se blacklist"
  2  MINVALUE 1 MAXVALUE 10000
  3  INCREMENT BY 1
  4  START WITH 1
  5  CACHE 20
  6  NOORDER CYCLE;

序列已创建。

SQL> CREATE OR REPLACE TRIGGER tr blacklist
  2  BEFORE INSERT ON blacklist
  3  FOR EACH ROW
  4  WHEN (NEW.ID IS NULL)
  5  BEGIN
  6  SELECT se blacklist.NEXTVAL INTO :NEW.ID FROM DUAL;
  7  END;
  8  /
触发器已创建
```

序列和触发器创建完成之后，可以使用 DESC 命令来查看创建的 blacklist 表结构信息，详细如图 20-8 所示。



图 20-8　blacklist 表结构图

20.3 功能实现

通过对系统的分析，银行卡电子交易系统可以简单地分为银行卡开户、用户激活、修改用户信息、删除用户、存款业务、取款业务、转账业务、交易记录查询、删除交易记录和黑名单添加等 10 个功能。下面将对这 10 项功能的操作进行详细讲解。

20.3.1 银行卡开户

银行卡开户是每个使用银行卡消费或者办理其他业务的用户必不可少的一项操作，通过开户每个用户可以获取一个银行卡。同样每个用户也可以办理多张银行卡，但是用户信息只需要录入一次即可。在进行开户时，用户需要向 users 和 card 两张表中插入数据，具体实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE new_user_card(
  2  uname VARCHAR2(10),
  3  usex VARCHAR2(4),
  4  uage NUMBER,
  5  uphone VARCHAR2(20),
  6  uaddress VARCHAR2(50),
  7  cnumber VARCHAR2(20)
  8  )
  9  IS
 10  BEGIN
 11  INSERT INTO users VALUES(uname,usex,uage,uphone,uaddress);
 12  INSERT INTO card VALUES(cnumber,'000000','0',2,(SELECT MAX(id) FROM
     users));
 13  END;
 14  /
过程已创建。
```

上述代码中，创建了名为 new_user_card 的存储过程，该过程用于接受用户插入的 6 个参数，然后执行 INSERT 语句，向 users 表和 card 表插入相应的数据。在向 card 表插入数据时，可以看出，默认情况下银行卡密码为 000000，剩余金额为 0.00 元，默认银行卡的状态为等待激活的。其中(SELECT MAX(id) FROM users)语句的作用是获取向 users 表中新插入的 ID 列值。

接下来则是调用该过程，因为创建完成过程之后不代表就已经执行了，需要调用该过程同时向该过程中传入相应的参数值，具体代码如下。

```
SQL> EXEC new_user_card('窗内网','男',6,'0398-00000000','河南郑州二七区
','111111111111');
```

```
PL/SQL 过程已成功完成。
```

到这里就已经成功地创建了一名用户，该用户的用户名为“窗内网”，银行卡号为111111111111以及其他用户信息内容。

20.3.2 用户激活

到这里银行卡开户就已经完成了，但是用户还不可以马上使用。因为新开户用户的银行卡默认为等待激活状态，需要对银行卡进行激活操作。在该系统中为银行卡的状态设置了激活和等待激活两种状态，其中1表示激活，2表示为激活。具体实现激活代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE activate card(
  2  cnumber VARCHAR2(20)
  3  )
  4  IS
  5  BEGIN
  6  UPDATE card SET cardtypeid=1 WHERE cardnumber=cnumber;
  7  END;
  8  /
过程已创建。
```

在过程中，结果是cnumber参数，该参数为银行卡号，管理员可以通过数据银行卡号对该银行卡进行激活，也就是将card表中的cardTYPEID列的值修改为1即可完成该项操作，以下代码就调用该过程。

```
SQL> EXEC activate card('111111111111');
PL/SQL 过程已成功完成。
```

完成手机卡激活后，用户可以通过使用SELECT语句来查看当前用户的状态以及其他的手机卡信息，代码如下。

```
SQL> SELECT * FROM card;
ID   cardNUMBER     cardPWD       cardMONEY       cardTYPEID      USERID
---  ----------     ---------     -------------   ------------    ---------
1     111111111111  000000        0               1               1
```

通过上述代码的运行结果可以看出，已经成功地将该银行卡进行了激活，cardTYPEID列值被修改为了1。

20.3.3 修改用户信息

当用户办理好银行卡开户业务之后，如果发现用户信息有不正确，或者需要修改个人用户密码，那么就可以通过修改用户信息功能进行操作。在该功能中又能细分为两个功能点：修改用户密码和修改用户基本信息等功能，下面将详细介绍这两个功能的实现。

1. 修改交易密码

用户在进行交易时需要使用交易密码，通过正确的交易密码方可进行交易操作。如果用户需要修改交易密码那么该功能可以满足要求。通常交易密码是和银行卡是绑定的，因此修改 card 表中的 cardPWD 列值即可，详细代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE update pwd(
  2  cnumber VARCHAR2(20),
  3  newpwd VARCHAR2(6)
  4  )
  5  IS
  6  BEGIN
  7  UPDATE card SET cardpwd=newpwd WHERE cardnumber=cnumber;
  8  END;
  9  /
过程已创建。
```

代码中，创建了 update_pwd 过程，在该过程中有两个参数，分别为卡号和新密码。用户只需要输入需要修改的卡号的修改的密码即可进行密码修改操作。以下代码执行该过程。

```
SQL> EXEC update pwd('111111111111','123456');
PL/SQL 过程已成功完成。
```

2. 修改基本信息

修改用户信息也是经常被用到的功能之一，可能用户的电话号码或者家庭住址发生了变化，那么通过该功能即可对用户的基本信息进行修改。该功能将会修改 users 表中数据信息，因为用户的基本信息存储在 users 表中。功能代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE update user(
  2  uname VARCHAR2(10),
  3  usex VARCHAR2(2),
  4  uage NUMBER,
  5  uphone VARCHAR2(20),
  6  uaddress VARCHAR2(50)
  7  )
  8  IS
  9  BEGIN
 10  UPDATE users SET usersex=usex,userage=uage,userphone=uphone,
 useraddress=uaddress
 11  WHERE username=uname;
 12  END;
 13  /
过程已创建。
```

代码中创建了名为 update_user 的存储过程，该过程主要用于修改 users 表字段值。以 USERNAME 列为条件进行修改。接下来就是运行该过程修改用户信息，代码如下。

20.3.7 转账业务

当A账户金额转移到B账户的过程称为转账，它的实现是将A账户金额减少而向B账户中添加。同样在程序中实现两个用户之间转账也是这样的实现思路。例如，现在新开户了一个账户222222222222，需要将11111111111账户中的钱为222222222222中转账200，那么实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE exchange(
  2  cnumber1 VARCHAR2(20),
  3  cnumber2 VARCHAR2(20),
  4  cmoney NUMBER
  5  )
  6  IS
  7  BEGIN
  8  UPDATE UPDATE card SET cradmoney=(SELECT cradmoney FROM card WHERE cardnumb
     er=cnumber1)-cmoney WHERE cardnumber=cnumber1;
  9  UPDATE UPDATE card SET cradmoney=(SELECT cradmoney FROM card WHERE cardnumb
     er=cnumber2)+cmoney WHERE cardnumber=cnumber2;
 10  INSERT INTO record VALUES((SELECT userid FROM card WHERE cardnumber=cnumber
     1),cmoney,TO DATE('2011/6/22','YYYY/MM/DD'),3);
 11  END;
 12  /
过程已创建。
```

上述操作中主要有3项对数据库的操作，第一将cnumber1用户金额减少，第二将cnumber2增加，第三向record表中插入记录数据。这样就实现了转账功能，调用过程代码如下。

```
SQL> EXEC exchange ('111111111111','222222222222'200);
PL/SQL 过程已成功完成。
```

20.3.8 交易记录查询

交易记录查询功能就比较简单了，用户通过使用SELECT语句即可查看到相关信息，但是record表中存在部分数据为外键ID，因此需要对外键进行过滤显示真实数据。代码如下。

```
SQL> SELECT u.username "用户名",
  2  r.recordmoney||'$' "交易金额",
  3  r.recordtime "交易时间",
  4  t.typename "交易类型"
  5  FROM users u,record r,types t
  6  WHERE r.typeid=t.id AND r.userid=u.id;
用户名        交易金额        交易时间        交易类型
```

```
---------- ---------   --------- -------------
窗内网        500        22-6 月 -11  存款业务
窗内网        100        22-6 月 -11  取款业务
窗内网        200        22-6 月 -11  转账业务
```

在前面共执行了三次交易，因此在这里查询出来 3 条记录分别为存款业务、取款业务和转账业务。如果用户再次进行交易，那么这里将会记录更多的交易记录。

20.3.9 删除交易记录

交易记录的删除是有管理员才可以进行的操作，因此该功能在使用时一定要注意。管理员可以通过数据用户名对该用户的所有交易记录进行删除，实现代码如下。

```
SQL> CREATE OR REPLACE PROCEDURE select record(
  2  uname VARCHAR2()
  3  )
  4  IS
  5  BEGIN
  6  DELETE FROM record WHERE userid=(SELECT id FROM users WHERE username=
     uname);
  7  END;
  8  /
过程已创建。
```

删除交易记录之前需要通过用户名获取用户 ID，然后通过 ID 作为删除条件进行删除操作，下面调用该过程。

```
SQL> EXEC select_record('窗内网');
PL/SQL 过程已成功完成。
```

20.3.10 黑名单添加

黑名单的添加是为了防止用户恶意地开户，或者其他违法行为的操作。如果用户违反了银行卡使用条款，那么则可以将该用户添加至黑名单中。当用户在黑名单中之后就无法再次办理银行卡业务，以及现在使用的所有银行卡将会全部冻结。实现黑名单的添加代码如下所示。

```
SQL> CREATE OR REPLACE PROCEDURE add blacklist(
  2  uname VARCHAR2(20),
  3  reason VARCHAR2(100)
  4  )
  5  IS
  6  BEGIN
  7  INSERT INTO blacklist VALUES((SELECT id FROM users WHERE username=uname),
  8  reason,TO_DATE('2011/6/22','YYYY/MM/DD'));
```

```
  9  END;
 10  /
```

执行该过程时，传入的参数是用户名称，而在 blacklist 表中为用户 ID，因此需要通过用户名查询 users 表中的 ID，然后插入到 blacklist 表中，接下来运行该过程代码如下。

```
SQL> EXEC select_record('窗内网','多次办理银行卡而不使用，经过核实属于恶意办理银行卡用户！');
PL/SQL 过程已成功完成。
```


20.4 数据库备份

数据库备份是对数据库的安全性的一个保障，如果在使用数据库期间发生重大事故导致数据库崩溃，可能数据库内的数据也会丢失。如果用户没有做数据库备份那么操作的损失是很明显的。如果用户出于数据库安全考虑做了数据库备份，那么用户即可通过备份文件对数据库的数据进行恢复，恢复后数据依然存在。

因此银行系统对用户数据备份是必不可少的功能，通常对数据的备份有很多种。最为常用的是使用 RMAN 工具对数据进行备份，除此之外还可以使用 Data Pump Export 工具对数据进行导入导出实现数据的备份。

20.4.1 备份数据库

RMAN 工具是 Oracle 数据库对数据备份和恢复定做的工具，通过使用该功能能够更加方便地对数据库进行备份和恢复。在使用该工具进行备份前需要对该工具进行简单的配置，首先分配通道，代码如下。

```
RMAN> RUN{
2> ALLOCATE CHANNEL banks_cd DEVICE TYPE DISK
3> FORMAT='D:\mydb\%u_%c';
4> BACKUP TABLESPACE banks CHANNEL banks_cd;
5> }
```

对数据库进行完全备份使用 BACKUP FULL 语句来完成，同时使用 TAG 参数和 FORMAT 参数来控制备份文件的位置以及备份文件的名称格式等信息，代码如下。

```
RMAN> RUN{
2> ALLOCATE CHANNEL banks_cd TYPE DISK;
3> BACKUP FULL
4> TAG full_db_backup FORMAT "D:\ mydb\db_t%t_s%s_p%p"(database);
5> RELEASE CHANNEL banks_cd;
6> }
```



20.4.2　导出数据库

在 Oracle 数据库中导出数据库会选择使用 Data Pump Export 工具，Data Pump Export 工具主要用于对数据和元数据进行转存到一个文件集的一组系统文件中。在使用该工具对数据库进行导出时需要创建外部目录，并且为操作用户赋予相应的操作权限。

```
SQL> CREATE DIRECTORY mybank AS 'D:\mydb';
目录已创建。

SQL> GRANT READ ON DIRECTORY mybank TO bankadmin;
授权成功。

SQL> GRANT WRITE ON DIRECTORY mybank TO bankadmin;
授权成功。
```

上述代码中，创建了名为 mybank 的目录，该目录用于存储导出数据文件的地址。然后为 bankadmin 用户赋予对该目录的读取和写入的权限。

权限和目录设置完成之后，就开始使用 Data Pump Export 工具进行导出操作。在进行操作之前需要在【运行】窗口中输入 CMD 命令，进入 CMD 命令窗口，然后在命令窗口进入到 Oracle 安装目录下的 BIN 目录，链接后的效果如图 20-9 所示。

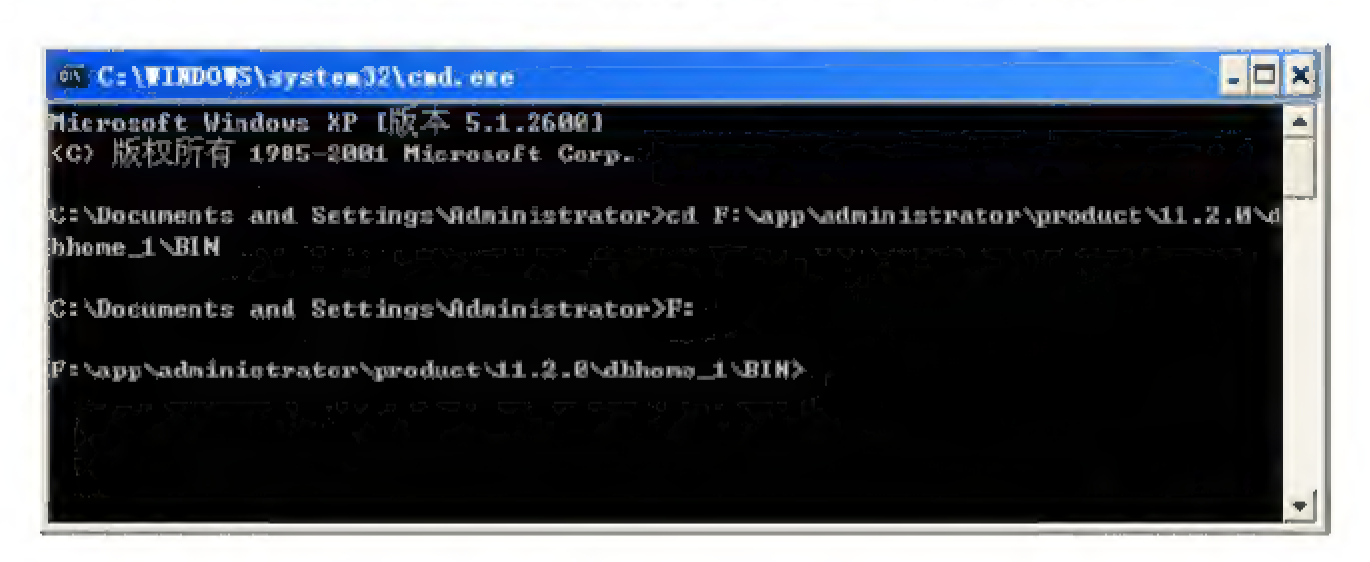


图 20-9　Data Pump Export 工具

打开工具之后，在命令行中输入以下代码对数据进行导出操作。

```
F:\app\administrator\product\11.2.0\dbhome 1\BIN>EXPDP bankadmin/000000
DIRECTOR
Y=mybank DUMPFILE=bfbank.dmp FULL=Y;
```

代码中使用 bankadmin 用户进行登录操作，通过 EXPDP 命令进行全库导出操作，FULL 表示全库导出的关键字。并且将导出的文件命名为 bfbank.dmp，运行以上代码输出内容如下。

```
F:\app\administrator\product\11.2.0\dbhome 1\BIN>EXPDP system/tiger
DIRECTORY=my
bank DUMPFILE=bfbank.dmp FULL=Y;
Export: Release 11.2.0.1.0 - Production on 星期四 8 月 25 18:31:05 2011
Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.
```

```
连接到: Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
启动 "SYSTEM"."SYS_EXPORT_FULL_01":  system/******** DIRECTORY=mybank
DUMPFILE=b
fbank.dmp FULL=Y;
正在使用 BLOCKS 方法进行估计...
处理对象类型 DATABASE_EXPORT/SCHEMA/TABLE/TABLE_DATA
使用 BLOCKS 方法的总估计: 141.8 MB
处理对象类型 DATABASE_EXPORT/TABLESPACE
处理对象类型 DATABASE_EXPORT/PROFILE
处理对象类型 DATABASE_EXPORT/SYS_USER/USER
处理对象类型 DATABASE_EXPORT/SCHEMA/USER
处理对象类型 DATABASE_EXPORT/ROLE
处理对象类型 DATABASE_EXPORT/GRANT/SYSTEM_GRANT/PROC_SYSTEM_GRANT
处理对象类型 DATABASE_EXPORT/SCHEMA/GRANT/SYSTEM_GRANT
处理对象类型 DATABASE_EXPORT/SCHEMA/ROLE_GRANT
处理对象类型 DATABASE_EXPORT/SCHEMA/DEFAULT_ROLE
处理对象类型 DATABASE_EXPORT/SCHEMA/TABLESPACE_QUOTA
处理对象类型 DATABASE_EXPORT/RESOURCE_COST
处理对象类型 DATABASE_EXPORT/TRUSTED_DB_LINK
处理对象类型 DATABASE_EXPORT/SCHEMA/SEQUENCE/SEQUENCE
处理对象类型 DATABASE_EXPORT/SCHEMA/SEQUENCE/GRANT/OWNER_GRANT/OBJECT_GRANT
处理对象类型 DATABASE_EXPORT/DIRECTORY/DIRECTORY
处理对象类型 DATABASE_EXPORT/DIRECTORY/GRANT/OWNER_GRANT/OBJECT_GRANT
......
```

通过运行结果输出信息可以看出，通过该命令导出的那些数据信息，由于数据结果代码过多，在这里省略了部分结果内容。此时在 D:\mydb 目录下会生成 BFBANK.DMP 的文件，详细如图 20-10 所示。

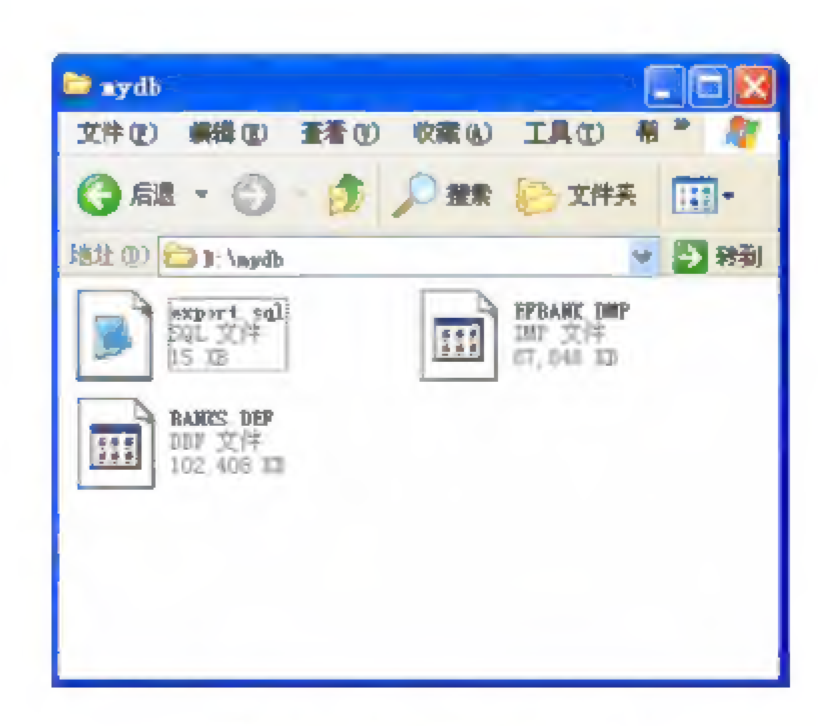


图 20-10　导出文件